LATVIJAS UNIVERSITĀTE
FIZIKAS UN MATEMĀTIKAS FAKULTĀTE
MATEMĀTIKAS NODAĻA

ATTĒLU APRAKSTĪŠANA AR KONVOLŪCIJU UN REKURENTAJIEM NEIRONU TĪKLIEM

MAĢISTRA DARBS

Autors: **Bruno Opermanis**

Stud. apl. bo10001

Darba vadītājs: prof. Dr. dat. Guntis Bārzdiņš

RĪGA 2017

Anotācija

Automātiskā attēlu aprakstīšana ir fundamentāla mākslīgās inteliģences problēma, kura apvieno kompjūter-redzes un naturālās valodas apstrādes algoritmus. Šajā darbā tiks apskatīta šī nozare, pielietots apbalvojumus izcīnījis modelis un pētītas šī modeļa variācijas. Attēlus aprakstošais modelis ir mākslīgais neironu tīkls, kurš sevī apvieno konvolūciju un rekurento neironu tīklu arhitektūras. Darbā vispirms abas arhitektūras ir apskatītas atsevišķi, kā arī teorija, uz kuru balstās modelis. Apskatītais modelis un tā variācijas tiek salīdzinātas, izmantojot klasiskos mašīnmācīšanās rādītājus un mašīntulkošanā izmantotas metrikas. Pēc dažiem rādītājiem oriģinālā modeļa uzlabojumi izrādījās lietderīgi. Modeļu apmācībā tika izmantots populārs mašīnmācīšanās rīks *Tensorflow* un programmēšanas valoda *Python*.

Atslēgas vārdi: Dziļie Neironu tīkli, Rekurentie Neironu tīkli, Automātiskā Attēlu aprakstīšana, Konvolūciju Neironu tīkli, Eksplodējošais/pazūdošais gradients, Reprezentāciju mācīšanās, *Tensorflow*

Abstract

Automatic image captioning is fundamental artificial inteligence problem which is a fusion of computer vision and natural language processing. In this work image captinioning field will be explored. Award winning image captioning model revisited and explored its variations. Image captioning model is an artifical neural network which consists of conolutional neural network and a recurrent neural network. In this work both these branches of architecture are studied theoretically and practically. Variations of the award winning image captioning model was compared with classical machine learning and machine translation metrics. According to some metrics the variations of the original model turned out usefull. Training of the models was done in popular machine learning tool *Tensorflow* and *Python* programming language.

Keywords: Deep Neural Networks, Recurrent Neural Networks, automatic image captioning, Convolutional Neural Networks, Exploding and Vanishing gradient, Representation learning, *Tensorflow*

Saturs

Apzīmējumi

$\mathbb{N}$ - naturālo skaitļu kopa,

$\mathbb{Z}$ - veselo skaitļu kopa,

$\mathbb{R}$ - reālo skaitļu kopa,

NN - neironu tīkls,

RNN - rekurentais neironu tīkls,

CNN - konvolūciju neironu tīkls,

GD - gradienta samazināšanas algoritms (*Gradient Descent algorithm*),

BP - atpakaļatgriezeniskais algoritms (*Backpropagation algorithm*),

BPTT - atpakaļatgriezeniskais algoritms laikā (*Backpropagation algorithm Through Time*),

CPU - *Central Processor Unit*,

GPU - *Graphic Processing Unit*,

$dim(\cdot)$ - matemātiska objekta dimensija,

$im(\cdot)$ - funkcijas attēla kopa,

$dom(\cdot)$ - funkcijas definīcijas kopa,

$(\cdot)$ - matemātisku objektu virkne,

$\{\cdot\}$ - matemātisku objektu kopa,

$\|\cdot\|$ - norma.

Ievads

Pirms apmēram 70 gadiem zinātnieki solīja, ka "domājošās" mašīnas drīz spēs tulkot, runāt, saprast attēlus saturošu informāciju, domāt gandrīz kā cilvēks. Diemžēl sajūsma drīz aprima, jo cilvēki saskārās ar neparedzētiem šķēršļiem šādu mašīnu un programmu izstrādāšanai.

Pēdējo gadu sasniegumi liek cilvēkos atgriezties tai pašai sajūsmai, kas bija pirms 70 gadiem, tikai daudzi solījumi jau tiek piepildīti. Mašīna ir uzvarējusi pasaules labāko spēles GO spēlētāju-cilvēku, automašīnas spēj pašas sevi vadīt, mašīnas spēj apstrādāt informāciju no naturālās valodas un attēliem aizvien līdzīgāk un līdzīgāk kā cilvēks.

Mākslīgās inteliģences sasniegumi saistāmi ar mašīnmācīšanās nozares izaugsmi, konkrētāk, ar dziļo neironu tīklu modeļu plašu pielietošanu.

Vairāki faktori ļauj dziļajiem neironu tīkliem paveikt to, kas iepriekš nebija praktiski iespējams. Lielāki skaitļošanas resursi - pēdējos gados neironu tīklu apmācīšanai tiek izmantotas GPU CPU vietā, kas var samazināt apmācības laiku vairākus desmitu reižu arī mājas apstākļos. Industriāliem mērķiem tiek izmantoti speciāli neironu tīkliem izstrādāti čipi - *tensor computing unit* (TPU). Taču dziļu neironu tīklu arhitektūras ir grūti apmācīt pazūdošā un eksplodējošā gradienta problēmas dēļ. Lai cīnītos ar šo problēmu tiek izmantotas jaunas neironu tīklu arhitektūras. Pateicoties Konvolūciju neironu tīkliem tika iegūti precīzākie modeļi objektu atpazīšanai attēlos. Šo arhitektūru iedvesmo bioloģisks novērojums ka dzīvnieku vizuālās uztveres neironi atbild par maziem uztveres lauka reģioniem. Ar jaunām rekurento tīklu arhitektūrām izstrādāti jauni ļoti kvalitatīvi modeļi, kuri risina naturālās valodas apstrādes problēmas kā tulkošana, teikumu ģenerēšana un analīze.

Šī nozare ir jauna un vairāki tās aspekti nav gana teorētiski izpētīti, daudz kas tiek balstīts uz heuristiskiem pieņēmumiem. Tāpēc šajā darbā liels uzsvars likts uz šo modeļu stipro un vājo pušu teorētisku un praktisku apskati.

Šā brīža Neironu tīklu popularitāte iedvesmoja autoru padziļināt̄āk apgūt šos modeļus, lai iegūtu izpratni, kā tie strādā, un, lai iegūtu intuīciju par to, kāpēc tie efektīvi risina sarežģītas mākslīgās inteliģences problēmas. Attēlu aprakstīšanas problemātika ir ļoti sarežģīta un apvieno divas fundamentālas mākslīgās inteliģences nozares - kompjūter-redzi un naturālās valodas apstrādi, tāpēc arī, lai izpētītu dziļos neironu tīklus, tika izraudzīta šī problemātika.

Darbā aprakstītais attēlu aprakstīšanas modelis tiek apmācīts bez speciāli sagatavotiem datiem, dati satur tikai attēlus un to aprakstošos teikumus, netiek izmantotas nekādas apriori sagatavotas teikumu formas. Šis modelis izmanto arī jaunu metodi - pārnesto mācīšanos, kad daļu no modeļa apmāca izolētam modelim, tad modeļa daļas saliek kopā un liek tam saprasties vienam ar otru.

Lai apmācītu attēlu aprakstīšanas modeli, tika lietots populārs dziļās mašīnmācīšanās rīks - *Tensorflow*, kuram ir programmējams interfeiss programmēšanas valodā *Python*. Darbā par pamatu tika ņemts 2015. gada MSCOCO attēlu aprakstīšanas sacensības uzvarējošā modeļa programmas kods, tas tika papildināts un modificēts, lai eksperimentētu ar jau esošo neironu tīkla arhitektūru. Oriģinālā modeļa variācijas tika salīdzinātas pēc vairākiem aspektiem.

1 Mākslīgie Neironu Tīkli

Mākslīgie neironu tīkli, jeb vienkārši neironu tīkli, ir skaitļošanas paradigma [1], kuru ir iedvesmojis veids kā strādā bioloģiskas nervu sistēmas. NN sastāv no liela skaita savstarpēji saistītu skaitļošanas vienību (neironu), kuras apvienotas veidā, kas risina kādu noteiktu problēmu. NN līdzīgi kā cilvēki mācās redzot piemērus. NN var tikt pielāgoti ļoti daudzām problēmām, piemēram,

- Objektu atpazīšanai attēlos,
- Regresijas un klasificēšanas problēmām,
- Naturālās runas apstrādei un ģenerēšanai,
- Kontroles sistēmās.

1.1. Neirons

Tiks sākts ar neirona definīciju, pēc tam izmantojot šo definīciju tiks aprakstīts neironu tīkls. Definīcijas ņemtas no mācību materiāla [2].

Definīcija 1. Neirons ir četrinieks (X,Y,σ,s), kur

- $X \subseteq \mathbb{R}^d, \quad Y \subseteq \mathbb{R}$ sauc par ieejas un izejas vērtību kopām, $X,Y \subseteq \mathbb{R}^d$,
- summēšanas funkciju $s : X \to \mathbb{R}$,
- aktivācijas $\sigma : \mathbb{R} \to Y$.

Kā arī $f = \sigma \circ s : X \to Y$ sauc par **neirona funkciju**.

Summēšanas funkcija, parasti, ir formā

$$s(x) = \sum_{i=1}^{d} w_i x_i + b,$$

kur $w_i \in R$ sauc par svariem un b sauc par novirzi. Dažreiz tiek lietotas neironu summēšanas funkcijas bez novirzes.

Aktivizācijas funkcija parasti ir nelineāra funkcija, piemēram,

- sigmoīda - $z \to 1/(1+e^z)$,
- identitātes - $z \to z$,
- pakāpiena - $z \to \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}$,
- hiperboliskā tangensa - $z \to 2/(1+e^{-2z}) - 1$,
- RELU (*rectified linear unit*) - $z \to \max(0,z)$,
- RELU6 - $z \to \min(\max(0,z),6)$,
- *softmax* aktivizācijas funkciju pielieto, kad nepieciešams modelēt sadalījuma masas funkciju pār diskrētu stāvokļu kopu $z = (z_1,....,z_K)$, tad *softmax* aktivizācijas funkciju definē kā

$$z \to \left(\frac{\exp(z_1)}{\sum_{k=1}^{K} \exp(z_k)},..., \frac{\exp(z_K)}{\sum_{k=1}^{K} \exp(z_k)} \right).$$

Pakāpiena aktivizācijas funkciju parasti neizmanto neironu tīklos, jo šīs aktivizācijas funkcijas atvasinājums ir 0 visur izņemot $z = 0$, līdz ar to nevarētu pielietot gradienta samazināšanas algoritmu. Taču pakāpiena funkcija bija pirmā aktivizācijas funkcija, kura tika pielietota neironu tīklos.

1.2. Neironu tīkls

Neironu tīkli šajā nodaļā tiks definēti un aprakstīti kā šajā mācību materiālā [2], šis apraksts un definīcija dod labu priekšstatu par vienvirziena neironu tīkliem. Taču šajā darbā tiks apskatīti arī neironu tīkli, kuri nedaudz "izkāpj" ārā no šīs definīcijas. Tā kā praksē un pētījumos neironu tīklu jēdziens tiek plaši pielietots, tad neironu tīklu jēdziens arī tiek pakāpeniski paplašināts.

Tā kā neironu tīkla definīcija ietver grafa jēdzienu, tad tiks atkārtota to definīcija, kā arī iepazīstināti turpmāk lietotie apzīmējumi.

Definīcija 2. Par **orientētu grafu** sauc divnieku $G = (V,E)$, kur G ir galīga, netukša kopa un $E \subseteq V \times V$. Parasti V sauc par virsotņu kopu un E par šķautņu kopu.

Tā kā V ir sanumurējama, tad šajā darbā lietošu pierakstu, kur V kopas elementi nav atšķirami no to indeksiem. Vēl tiks definētas dažas kopas, kuras mums palīdzēs strādāt ar grafiem:

- Kopa $P(j) := \{i \in V : (i,j) \in E\}$ tiks saukta par j virsotnes **tiešo priekšteču kopu**. Tāpat $S(j) := \{i \in V : (j,i) \in E\}$ sauc par **tiešo pēcteču kopu**. Ja $S(j) = \emptyset$, tad j sauc par **noteci**. Ja $P(j) = \emptyset$, tad j sauc par **avotu**.
- Virsotņu virkni $(j_0,....,j_l) \in V$ sauc par **ceļu** ar garumu l, ja eksistē šķautnes $e_i = (j_{i-1},j_i), i = 1,...,l$. Ceļu sauc par **ciklu**, ja $j_0 = j_l$.

Tagad ir definēts viss vajadzīgais, lai definētu vienvirziena neironu tīklu.

Definīcija 3. Par **neironu tīklu** sauc četrinieku $M = (X,Y,G,N)$, kur G ir orientēts grafs bez cikliem un $N = \{n^1,....,n^D\}$ ir neironu $n^j = (X^j,Y^j,\sigma^j,s^j)$ kopa, kur j-tais neirons asociēts ar virsotni $j \in G$ un

$$X^j = \begin{cases} \bigtimes_{u \in P(j)} Y^u & , P(j) \neq \emptyset \\ X & , P(j) = \emptyset \end{cases}$$

kur X ir tīkla funkcijas F (kura tūlīt tiks definēta) ieejas kopa.

Lai definētu F, vispirms tiek ieviestas palīgfunkcijas $F^j : X \to Y^j, \quad j = 1,...,D$, kuras definē rekursīvi

$$F^j(x) = \begin{cases} f^j(F^{u_1}(x),...,F_{u^{|P(j)|}}(x)) & , \text{ja} P(j) \neq \emptyset \\ f^j(x) & , \text{ja} P(j) = \emptyset \end{cases},$$

kur $\{u_1,...,u_{|P(j)|}\} = P(j)$. Tad tīkla funkciju tiek definēta kā

$$F(x) = \begin{pmatrix} F^{j_1}(x) \\ ... \\ F^{j_k}(x) \end{pmatrix}, \quad \text{kur} \quad S(j_1),...,S(j_k) = \emptyset$$

jeb $j_1,...,j_k$ ir grafa G noteces.

Tikko dotajā definīcijā tika definēta arī G apakšgrafu funkcijas F^j, kuras ir izrēķināmas, jo G ir galīgs un nesatur ciklus.

Turpmāk šajā darbā, ja tas neizraisīs divdomības, tad ar neironu tiks saprasts gan G virsotne, gan neirona funkcija. Kā arī neironu tīkls tiks saukts arī par vienkārši tīklu vai par modeli.

1.3. Neironu tīklu topoloģijas

Par neironu tīklu topoloģijām jeb arhitektūrām sauc neironu tīklu grafu saimes.

Lai atvieglotu darbu ar neironu tīklu arhitektūrām, neironus parasti grupē **slāņos**. Starp vienā slānī esošiem neironiem neeksistē šķautnes. Neironu grupēšana slāņos ļauj daļēji pārrakstīt tīkla funkciju ar matricu palīdzību, un praksē ļauj izmantot augsti optimizētus lineārās algebras algoritmus, lai lietotu un novērtētu tīklu funkcijas.

Neironu tīklu arhitektūras iedalās divās grupās, balstoties uz to grafu G īpašībām:

- vienvirziena tīkli - respektīvajos grafos nav ciklu;
- rekursīvie tīkli - respektīvie grafi satur ciklus.

1.3.1. Apmācības problemātika

Tagad tiks apskatīta patvaļīga NN arhitektūru, neironu iedalījumi slāņos tiek ignorēti. Šajā darbā ar jēdzieniem 'novērtēt' un 'apmācīt' tiks saprasts viens un tas pats, jēdziens 'novērtēt' vairāk saistāms ar matemātiskās statistikas nozari un 'apmācīt' ar mašīnmācīšānās nozari, kura ir mākslīgās inteliģences apakšnozare.

Apmācības mērķis ir aproksimēt kaut kādu nepārtrauktu funkciju $\mathbb{F} : X \to Y$, kuras aprēķināšanas algoritms nav dots, bet ir novērota n reizes

$$t_i = \mathbb{F}(x_i) + \epsilon_i, \quad i = 1,...,n$$

ar neironu tīkla funkciju F, pielāgojot svarus neironu parametrus. Ar ϵ_i tiks apzīmēts nenovirzīts ($E\epsilon_i = 0$) baltais troksnis jeb datos nav sistemātiskas novirzes no $\mathbb{F}$. Neironu tīklus pielieto arī nepiesārņotu datu gadījumos, bet tos var reducēt uz tikko definēto problēmu, kad $\forall i, \epsilon_i = 0$.

Apmācības uzdevums ir atrast tādus F parametrus w^*, kuri minimizē kaut kādu funkcionāli, kurš raksturo 'attālumu', starp F un $\mathbb{F}$ kopas X un Y novērotajās apakškopās ar datu pāriem $(x_i,t_i),i = 1,...,n$. Kā funkcionāli optimizēšanai lieto **izlases kļūdas funkciju**

$$\hat{E} = \frac{1}{n}\sum_{i=1}^{n} E_i = \frac{1}{n}\sum_{i=1}^{n} e\left(t_i,F(x_i)\right),$$

kur e ir tā saucamā kļūdas jeb zaudējumu funkcija, kurai izpildās

- nenegativitāte $\forall x_1,x_2 : \quad e(x_1,x_2) > 0$,
- $e(x_1,x_2) = 0 \iff x_1 = x_2$.

Šajā nodaļā strādāsim ar kļūdu funkciju $e(x_1,x_2) = \|x_1 - x_2\|^2$, taču praksē izmanto arī daudzas citas kļūdu funkcijas, piemēram:

- **kros-entropijas** kļūdas funkciju

  $$e(p_1,p_2) = -\sum_{x} p_1(x)\log p_2(x),$$

  kur p_1,p_2 ir sadalījumu masas funkcijas, kros-entropijas kļūdas funkciju lieto, ja Y ir diskrēta un nesakārtota kopa, piemēram, klasifikācijas situācijās
- tikko minēto $\|x_1 - x_2\|^2$ bieži izmanto, ja Y ir skaitļu kopa (veselo, naturālo vai reālo), vai skaitļu vektoru kopa.

Tālāk apskatīsim kā $\hat{E}$ tiek minimizēta ar populārāko algoritmu Neironu tīklu apmācīšanai - Backpropogation algoritmu.

1.4. Tīkla novērtēšanas algoritms

Populārākais algoritms neironu tīkla novērtēšanai ir atpakaļatgriezeniskais (*backpropogation*) algoritms. Šajā nodaļā tiks aprakstīts vienkāršākais gadījums, praksē parasti tiek lietotas dažādas šī algoritma modifikācijas, kuras tiks pieminētas šīs nodaļas beigās.

Tā kā atpakaļatgriezeniskais algoritms ir GD speciāls gadījums, kur GD tiek pielietots, lai novērtētu neirona tīkla parametrus dotai datu kopai $\{(x_1,t_1),...,(x_n,t_n)\}$, kur $x_i \in X$ un $t_i \in Y$, tad tiks apskatīta GD algoritma galvenā ideju, jo tā ļaus gūt labāku izpratni par BP vājajām un stiprajām pusēm. Turpmāk visus neironu tīkla parametri tiks sakārtoti

vienā vektorā W, kad tiks specifiski runāts par kādu neirona parametru vektoru, tad tiks lietoti apzīmējumi w^j,b^j vektorus.

1.4.1. Gradienta samazināšanās algoritms

GD algoritmu pielieto gludas funkcijas $E = E(w)$ optimizācijai, bet praksē tiek lietots arī tikai nepārtrauktām funkcijām. Tā kā maksimizāciju var viegli reducēt uz minimizēšanu, tad tiks apskatīta tikai minimizēšana. Tiks pieņemts, ka argumentu (neironu tīkla kļūdu funkcijas gadījumā tie būs parametri) kopa ir $\mathbb{R}^d$, $w \in \mathbb{R}^d$. Šis gradienta samazināšanās algoritma apraksts balstīts uz lekciju materiālu *grad*.

Definīcija 4. Par diferencējamas funkcijas E gradientu sauc vektoru, kurš satur visus E parciālatvasinājumus

$$\nabla E := \left(\frac{\partial E}{\partial w^1}, ..., \frac{\partial E}{\partial w^d} \right).$$

Būtībā gradients ir vektor-funkcija, taču šajā darbā bieži ar gradientu tiks domāts šīs funkcijas vērtība dotai ieejai, tātad vienkārši vektors.

Ļoti noderīga un plaši izmantota gradienta īpašība ir tā, ka gradients pie dota argumenta w_0 norāda uz virzienu kādā funkcija E visātrāk pieaug no punkta w_0, citiem vārdiem sakot

$$\nabla E(w_0) = \arg\max_{\Delta w} \lim_{\epsilon \to 0} E(w_0 + \epsilon \Delta w),$$

pieņemot, ka funkcijai E eksistē atvasinājums no visam pusēm.

Šī gradienta īpašība šajā optimizācijas kontekstā ir ļoti nozīmīga, jo tā parāda, kādā virzienā izmainīt argumentu w_0, lai samazinātu (par $-\nabla E$) funkcijas E vērtību.

Sekojošais apgavlojums definē GD algoritmu.

Apgalvojums 1. *Eksistē tāds skaitlis $\lambda > 0$, pie kura GD algoritms aproksimē gludas funkcijas E lokālu minimumu E_{min} vai seglu punktu pēc patikas tuvu, iteratīvi pārstaigājot argumentu w telpu pēc likuma*

$$w_{t+1} := w_t - \lambda \nabla E(w_t),$$

ar patvaļīgu w_0.

Praksē w_t nonāk seglu punktā ļoti reti. GD algoritms darbojas polinomiālā laikā pēc argumentu skaita d. Pilnā pārlase (pieņemsim, ka $\mathbb{R}^d$ tiktu reducēta uz diskretizētu hiper-

kubu) būtu eksponenciāla laika algoritms pēc d meklējot globālu minimumu, un nedeterministiski polinomiāls meklējot lokālu optimimumu.

Kā galvenie GD algoritma negatīvie aspekti ir šādi:

- jāizvēlas skaitlis λ, pārāk mazām λ vērtībām w_t lēni konverģētu uz lokālo minimumu, taču pie pārāk lielām λ vērtībām w_t var pat diverģēt vai oscilēt;
- tiek atrasts tikai lokālais minimums nevis globālais (piemēram, pretēji atbalsta vektoru mašīnām, kura apmācības algoritms garantē globālu optimumu).

Svarīgi piebilst, ka ne visi neironu tīkli ir diferencējami visā parametru telpā. Piemēram, gadījumos kad tiek lietotas RELU aktivizācijas funkcijas. Tādos gadījumos tik un tā sekmīgi tiek pielietoti uz GD balstīti algoritmi kā atpakaļatgriezensikais algoritms.

1.5. Atpakaļatgriezeniskais algoritms

Tā kā GD algoritmam nepieciešams kļūdas funkcijas gradients, tad tagad tuvāk apskatīsim $\hat{E}$ atvasinājumu ar $e(x,y) = \sum_{k=1}^{d}(x_k - y_k)^2$ (x un y ir vektori). Šis atpakaļatgriezeniskā algoritma apraksts balstīts uz [3].

Apskatīsim tuvāk, kā ar BP algoritmu tiek aprēķināts gradients, iegūtais priekšstats noderēs tālākās nodaļās, kad tiks apskatītas sarežģītākas NN arhitektūras.

Tā kā izlases kļūdu funkcija ir kompleksa funkciju kompozīcija, tad nozīmīgu lomu šajā nodaļā spēlēs funkciju kompozīcijas atvasinājuma likums, jeb *ķēdes likums*

$$\frac{dg(f(x))}{dx} = \frac{dg(y)}{dy}\frac{df(x)}{dx}, \quad \text{kur} \quad y = f(x).$$

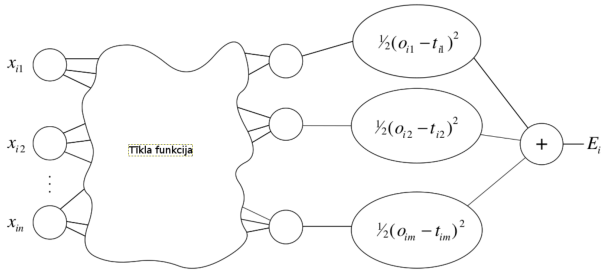
Ķēdes likums ir ļoti noderīgs aprēķinot dziļu funkciju kompozīciju atvasinājumus. Piemēram, ja

$$f_1 \circ ... \circ f_k \circ ... \circ f_m,$$

tad funkcijas $f_k : im(y_{k+1}) \times \theta \to dom(y_{k-1})$ atvasinājums pēc θ būs atkarīgs tikai no funkciju f_1,...,f_{k-1} atvasinājumiem. Drīz tiks parādīts, kā šī īpašība palīdz BP kontekstā.

Atvasinām $\hat{E}$ līdz tīkla funkcijai pēc patvaļīga tīkla parametra w_h

$$\frac{\partial \hat{E}(W)}{\partial w_h} = \frac{1}{n}\sum_{i=1}^{n} 2\left(F(x_i) - t_i\right)\frac{\partial F}{\partial w_h}(x_i).$$



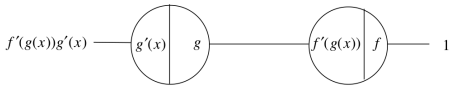
1.1. att. Kļūdas funkcijas atkarības no tīkla funkcijas ilustrācija

Aprēķināt F parciāl-atvasinājumus nav triviāls uzdevums, jo tā ir patvaļīgi dziļa funkciju kompozīcija 1.1. Tagad tiks tuvāk apskatīts $\frac{\partial F}{\partial w_h}(x_i)$, x_i aizstājot ar mainīgo $x \in X$.

Neironu tīkla gradienta aprēķināšana

Vispirms tiks apskatīts, kā atvasināšana izpildās 2 neironiem to iespējamajās savstarpējās situācijās grafā G, pēc tam iegūtie likumi tiks vispārināti grafam G.

- **Neironi f un g atrodas virknē.**



1.2. att. Atvasinājums, ja neironi atrodas virknē

Šajā gadījumā izpildās atvasināšanas likums funkciju kompozīcijas gadījumā 1.2.

- **Neironi f_0 un f_{m+1} atrodas uz viena ceļa grafā G, taču starp tiem atrodas vairāki citi neironi f_1,...,f_m,** jeb

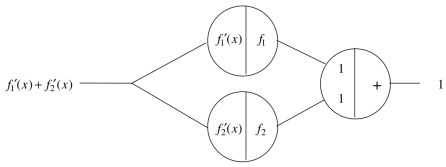
$$f_\theta^0 \circ ... \circ f_{\theta_1}^1 \circ ... \circ f_{\theta_k}^k \circ ... \circ f_m,$$

kur $f_{\theta_k}^k : im(f^{k+1}) \to im(f^k)$ ir kondicionēts uz f_k pārējiem argumentiem (pārējie parametri ir fiksēti). Šajā gadījumā var pielietot ķēdes likumu katrām 2 funkcijām

$$\frac{\partial f_{\theta_k}^k \circ f_{\theta_{k+1}}^{k+1}}{\partial x_{k+1}} = \frac{\partial f_{\theta_k}^k}{\partial x_k} \cdot \frac{\partial f_{\theta_{k+1}}^{k+1}}{\partial x_{k+1}},$$

kur $x_{k+1} \in dom(f^{k+1}_{\theta_{k+1}})$ un $x_k \in dom(f^k_{\theta_k})$. Tad garās funkciju kompozīcijas atvasinājums ir garš funkciju reizinājums.

- Neironiem g un f ir kopīgs pirmstecis 1.3.



1.3. att. Atvasinājums, ja neironi atrodas paralēli

Var redzēt, ka tad šo neironu atvasinājumi vienkārši saskaitās.

Kā arī individuāla neirona $f^j(x^j) = \sigma\left(\sum_{i=1}^{d_j} w^{ij}x^{ij} + b^j\right)$ atvasinājumi pēc parametriem un argumentiem ir:

$$\begin{aligned}
\frac{\partial f^j}{\partial x^{ij}}(x^j) &= \sigma'(x^j)w^{ij}, \\
\frac{\partial f^j}{\partial w^{ij}}(x^j) &= \sigma'(x^j)x^{ij}, \\
\frac{\partial f^j}{\partial b^j}(x^j) &= \sigma'(x^j).
\end{aligned}$$

Pirms aprakstam BP algoritmu tiks definēti daži objekti.

- J^j ir kopa, kas satur visus neirona n^j pēcteču neironu indeksus.
- $y^j := f^j(x^j)$.

BP algoritms: Atpakaļatgriezenkais atlgoritms vēturiski tiek sadalīts divās daļās - uz priekšu padeves fāzē un atpakaļatgriezeniskajā fāzē. Tad, lai aprēķinātu F gradientu dotai tīkla ieejai x tiek veiktas sekojošas darbības.

- **Uz priekšu padeves fāze:** katram ceļam grafā G secīgi no avotiem līdz notecēm aprēķina:
  - neironu vērtības $y^j = f^j(x^j)$;

– individuālu neironu parciālo atvasinājumu vērtības

$$\frac{\partial f^j}{\partial x_i^j}(x^j), \quad \frac{\partial f^j}{\partial w_i^j}(x^j), \quad \frac{\partial f^j}{\partial b^j}(x^j), \quad j = 1,....,D.$$

Kad šī fāze tiek skaitļota, šīs vērtības tiek saglabātas atmiņā kopā ar neironu ieeju vērtībām x^j, lai tās varētu izmantot atpakaļatgriezeniskajā fāzē.

- *Atpakaļatgriezeniskā fāze:* izmantojot ķēdes likumu un iepriekš apskatītās neironu attiecības grafā G parciālatvasinājumus, pēc parametriem var aprēķināt pēc likuma

$$\frac{\partial F}{\partial w_i^j}(x) = \frac{\partial f^j}{\partial w_i^j}(x^j) \sum_{v \in J^j} \frac{\partial f^v}{\partial y^j},$$

kur $\frac{\partial f^j}{\partial w_i^j}(x^j) = \sigma'(x^j)x_i^j$ skaitliskās vērtības jau tika aprēķinātas uz priekšu padeves fāzē.

Atpakaļatgriezeniskā algoritma labākai izpratnei tas tika implementēts tīkla arhitektūrai ar 2 pilnsaistes slāņiem valodā *Python* un izmēģināts simulētai klasifikācijas problēmai. Implementācijā neironi tika reprezentēti ar atsevišķām funkcijām, nevis izmantota matricu forma [19].

Vēl jāpiebilst, ka šajā darbā tiks apskatītas neironu tīklu arhitektūras, kurām būs nepieciešami atpakaļatgriezeniskā algoritma speciālas modifikācijas.

1.5.1. Atpakaļatgriezeniskā algoritma paveidi

Lai uzlabotu algoritma konverģenci, tiek lietotas dažādas heuristikas. Zemāk pieminētas dažas no tām.

- Dažādas mācību ātruma λ samazināšanas stratēģijas. Parasti kā nosacījums, lai λ samazinātos ir iterāciju skaits vai relatīvi lielas kļūdu funkcijas vērtību svārstības.

- Lai izvairītos no gradienta kaitīgām svārstībām, parastā GD vienādojuma vietā lieto

$$w_{t+1} := w_t - \lambda \nabla E(w_t) - \alpha \nabla E(w_{t-1}),$$

kur tiek izmantota gradientu vērtība no iepriekšējā laika soļa $t-1$, tādā veidā piešķirot virknei (w_t) "inerci".

- Pēdējos gados īpašu popularitāti ieguvis Stohastiskais gradienta samazināšanās algoritms, kad gradients katrā iterācijā netiek aprēķināts uz visiem datu punktiem, bet tikai uz gadījuma apakšizlases. Tādā veidā gan padarot iterācijas ātrākas (gradients netiek aprēķināts visos datu punktos), gan arī palīdz cīnīties ar pārpielāgošanas (*overfitting*) problēmu.

1.6. Dziļo neironu tīklu apmācība

Ar dziļajiem neironu tīkliem tiek saprasti neironu tīkli ar 'dziļām' neironu funkciju kompozīcijām. Šis termins nav strikti definēts.

Apmācot dziļos neironu tīklus jau pirms vairākiem gadiem tika identificēta problēmas - sarūkošā un eksplodējošā gradienta problēmas. Kā jau problēmu nosaukumi liecina - gradientu vērtības kļūst ļoti mazas vai tieši pretēji - ļoti lielas. Līdz ar to apmācība kļūst ļoti lēna vai pat apstājas sarūkošā gradienta gadījumā, vai arī virknes (w_t) elementi oscilē tik spēcīgi, ka diverģē no w^*, eksplodējošā gradienta gadījumā.

Pētnieki ir atzinuši, ka liela nozīme šīs problēmas esamībai ir aktivizācijas funkcijas izvēlei. Visbiežāk izmantoto aktivizācijas funkciju, sigmoīda un hiperboliskā tangensa, atvasinājumi nav lielāki par respektīvi 1/4 un 1. Līdz ar to - reizinot lielumus mazākus par 1 daudzas reizes, kā tas notiek atpakaļatgriezeniskajā algoritmā dziļiem neironu tīkliem, kļēdes likuma dēļ

$$\frac{\partial F}{\partial w_i^j}(x) = \sigma'(x^j)x^{ij}\left(\sum_{v\in J^j}\sigma'(x^v)w^{vj}\right)$$

parametru atvasinājumi pieņem ļoti mazas vērtības.

Skaidrs, ka, ja izvēlētos aktivizācijas funkciju ar atvasinājuma vērtībām, kuras ir lielākas par 1 "lielā" nulles punkta apkārtnē eksplodējošā gradienta problēma kļūtu aktuāla.

Tāpēc ir svarīgi, ka atvasinājums ir tieši 1 nulles punkta apkārtnē. Ja aktivizācijas atvasinājums būtu 1 visiem $z \in \mathbb{R}$ aktivizācijas funkcija būtu vienkārši identitātes funkcija, līdz ar to neironu tīkls kļūtu par lineāru modeli, līdz ar to šis risinājums arī nav vēlams.

RELU aktivizācijas funkcija ir labs šīs problēmas risinājums, jo tās atvasinājums $\forall z > 0$ ir 1, un lai modelis nekļūtu lineārs $\forall z < 0$ vienkārši $RELU(z) = 0$.

2 Konvolūciju neironu tīkli

Konvolūciju tīkli ir vienvirziena NN. Šajos tīklos visi neironi tiek organizēti slāņos. Kopā ar pilnsaistes slāņiem tiek lietoti arī konvolūciju un apvienošanas (*pooling*) slāņi. Literatūrā var sastapties arī ar gadījumiem, kad aktivizācijas funkcijas tiek atšķirtas no summācijas funkcijas kā atsevišķs slānis.

2.1. Konvolūciju slāņa motivācija

Pilnsaistes tīkli nav labi piemēroti ieejas vektoriem ar ļoti lielu dimensiju, piemēram, kā tas ir ar attēliem. Ja pielietotu pilnsaistes neironu tīklu melnbaltam attēlam ar izšķirtspēju 300×300 pikseļu, tad katram neironam tīkla pirmajā slānī būtu 90000 parametru (neņemot vērā novirzes parametrus). Tas ne tikai palielinātu nepieciešamo skaitļošanas resursu daudzumu, bet arī palielinātu pārpielāgošanas risku.

Līdz ar to tikko minētajai problēmai ir nepieciešams elegantāks risinājums, kas samazinātu gan parametru skaitu pirmajā slānī, gan palīdzētu tīklam labi ģeneralizēt attēlā redzamos objektus.

2.2. Funkciju konvolūcijas

Definīcija 5. Pieņem, ka f un g ir Rīmaņa mērāmas funkcijas, tad, ja visiem $x \in R$ eksistē Rīmaņa integrālis

$$\int_{-\infty}^{\infty} f(\tau)g(x-\tau)d\tau,$$

tad to sauc par funkciju f un g konvolūciju un apzīmē ar $f * g$.

Parasti g sauc par konvolūcijas kodolu, un f par konvolvējamo funkciju, lai arī skaitliski nav svarīgi, kura funkcija ir kodols un kura konvolvējamā, ja funkcijas integrālī samaina vietām, iegūst to pašu vērtību.

Kad funkcijas f un g pieņem diskrētus argumentus, tad integrāli aizstāj ar summas zīmi, tādas konvolūcijas sauc par diskrētajām konvolūcijām. Kā arī bieži praksē konvolūcijas izmanto galīgos domēnos. Vēl jāpiebilst, ka, ja kodols g ir simetrisks pret $t = 0$, tad var rakstīt $f * g(x) = \int_{-\infty}^{\infty} f(\tau)g(t + \tau)d\tau$.

Bieži konvolūcijas tiek izmantotas arī gadījumos, kad funkciju domēni X ir daudzdimensionāli. Tad konvolūciju definīcija arī krasi neatšķiras

$$(f * g)(x_1,...,x_n) = \int_{-\infty}^{\infty} ... \int_{-\infty}^{\infty} f(\tau_1,...,\tau_n)g(x_1 - \tau,...,x_n - \tau_n)d\tau_1...d\tau_n.$$

Konvolūcijas jēdziens tiek izmantots vairākos matemātikas pielietojumos, piemēram, šādos:

- varbūtību teorijā gadījumu lielumu summas sadalījuma funkcijas iegūšanai;
- matemātiskajā statistikā slīdošās vidējās vērtības modelis ir diskrēta konvolūcija;
- attēlu izšķirtspējas uzlabošanai pielieto dekonvolūciju metodes, kurās konvolūcijas kodols tiek pieņemts kā "pludinātājs/piesārņotājs" (bieži Gausa sadalījuma blīvuma funkcija), no $f * g$ iegūst asu attēlu f.

Vēl jāpiebilst, ka konvolūcijas definīcija fiksētam x ir identiska skalārā lieluma definīcijai starp funkcijām g un $h(\tau) = f(x - \tau)$. Līdz ar to konvolūcijai izpildās skalārā reizinājuma īpašība - jo funkcija f ir kodolam g "līdzīgāka pēc formas", jo konvolūcija sasniegs lielāku vērtību.

2.3. Attēla konvolūcija

Neironu tīklu kontekstā konvolūcijas slāņa ideja ir izmantot telpisko informāciju starp pikseļiem attēlā. Lai to izdarītu, šie tīkli izmanto diskrētas konvolūcijas.

Lai arī konvolūciju tīkli var tikt pielietoti krāsainiem RGB (sarkans/zaļš/zils) attēliem, vienkāršības labad tiks apskatīts melnbaltu attēlu gadījums.

Lai apskatītu specifiskāk diskrēto konvolūciju, tiek definēts melnbalts attēls ar izšķirtspēju $n_1 \times n_2$ kā attēlojums

$$I : \{1,...,m_1\} \times \{1,...,m_2\} \to [0,1]\,,$$

attēlojums $I(i,j)$ piekārto pikselim ar indeksiem i,j tā vērtību - ja 1-tad balts, ja 0-tad melns, savādāk kāds no pelēkās krāsas toņiem.

Melnbaltajā gadījumā konvolūciju izprašanā var labāk lietot intuīciju, jo tad attēlu var iztēloties kā virsmu 3-dimensionālā telpā.

Dotam konvolūciju kodolam $K : \mathbb{Z}^{2h_1+1\times 2h_2+1} \to R$ attēla I konvolūcija tiek pierakstīta kā

$$(I * K)(r,s) = \sum_{u=-h_1}^{h_1} \sum_{v=-h_2}^{h_2} K(u,h)I(r+u,s+v).$$

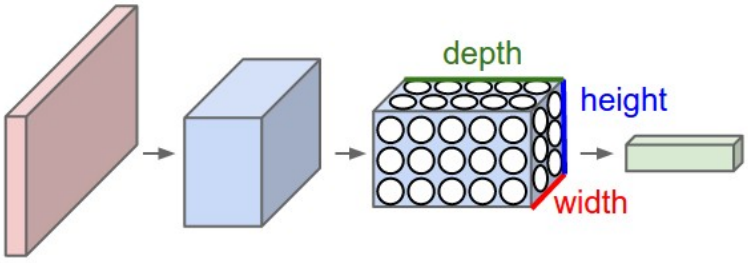
Lai tikko pierakstīto konvolūciju varētu definēt visiem $(r,s) \in \{1,...,m_1\} \times \{1,...,m_2\}$, nepieciešams attēla I vietā lietot

$$I'(i,j) = \begin{cases} I(i,j) & , \quad (i,j) \in \{1,...,m_1\} \times \{1,...,m_2\}, \\ 0, & \quad (i,j) \notin \{1,...,m_1\} \times \{1,...,m_2\} \end{cases} .$$

2.4. Konvolūcijas slānis

Tagad konvolūciju slānis tiks tīri definēts ar konvolūciju palīdzību, pēc tam tas tiks izrakstīts neironu formā. Šis apraksts balstīts uz [12]. Tiek pieņemts, ka slāņi tiek indeksēti ar l.

Konvolūciju slāņa ieejas elementi var būt daudz-dimensionāli tenzori, bet, tā kā šī nodaļa tikai apskata galvenās idejas, tad tiks apskatīts gadījumus, kad konvolūciju slāņu izejas un ieejas tenzoriem ir 3 dimensijas - augstuma/platuma/dziļuma dimensijas 2.1.



2.1. att. Konvolūciju slāņa (gaiši zilā) pielietošana attēlu reprezentējošajam tenzoram (rozā)

l-tā slāņa izejas tenzora dimensijas tiek apzīmētas ar $m_1^l/m_2^l/m_3^l$. Tad, ja tīkla l-tais

slānis ir konvolūciju slānis, tad to var piekrastīt šādi

$$Z_i^l(r,s) = \sigma(B_i^l + \sum_{j=1}^{m_3^{l-1}} (K_{i,j}^l * Y_j^{l-1})(r,s)), \quad i \in \{1,...,m_3^l\},$$

kur σ ir aktivācijas funkcija, B_i^l ir noviržu parametru matrica. Tad l-tā slāņa izejas tenzoram atkal ir 3 dimensijas, $(r,s,i) \in \{1,...,m_1^l\} \times \{1,...,m_2^l\} \times \{1,...,m_3^l\}$.

Novērtētās konvolūcijas kodoli $K_{i,j}^l$ strādā kā "trafareti", kuri tiek "slidināti" pa attēlu (vai kādu slēpto slāni). Jo attēla fragments $I'(r',s'), \quad (r',s') \in \{r - h_1, s + h_1\} \times \{r - h_2, s + h_2\}$ vairāk līdzinās "trafaretam", jo konvolūcija sasniedz lielāku vērtību (iepriekš minētā skalārā reizinājuma īpašība). Līdz ar to konvolūcija $K_{i,j}^l * I'$ uzrāda/detektē, kur attēls lokāli ir līdzīgs kodolam $K_{i,j}^l$. Šādā veidā konvolūciju tīkls izmanto attēlu telpisko informāciju.

Lai slāni pārrakstītu neironu formā, katrs neirons ir jāpielieto vairākiem slāņa ieejas konfigurācijām. Ar j tiks indeksēta tenzora dziļuma dimensija.

Tagad konvolūciju slānis tiks izrakstīts kā neironu summācijas funkcijas,

$$\begin{aligned}
Z_i^l(r,s) &= B_i^l + \sum_{j=1}^{m_3^l} (K_{i,j}^l * Y_j^{l-1})(r,s) \\
&= \sum_{j=1}^{m_3^l} \sum_{u=-h_1}^{h_1} \sum_{v=-h_2}^{h_2} K_{i,j}^l(u,v) Y_j^{l-1}(r+u, s+v) + B_i^l \\
&= \sum_{j=1}^{m_3^l} \sum_{u=-h_1}^{h_1} \sum_{v=-h_2}^{h_2} w_{u,v}^{i,j,l} Y_j^{l-1}(r+u, s+v) + B_i^l \\
&= \sum_{j=1}^{m_3^l} z_{i,j}^{l-1}(Y_j^{l-1}(r,s)),
\end{aligned}$$

kur $z_{i,j}^{l-1}$ ir neirona $n_{i,j}^l$ summācijas funkcija $Y_J^{l-1}(r,s)$. Summācijas funkcijas parametri ir $w_{u,v}^{i,j,l}, \quad (u,v) \in \{-h_1,...,h_1\} \times \{-h_2,...,h_2\}$ un $b_{i,j}^l = B_i^l / m_3^l$. Var redzēt, ka, lai pārrakstītu neirona parametrus, nepieciešams organizēt matricā, un katram neironam ir 3 indeksi - i,j,l. Pilnu neironu funkciju iegūst pēc aktivizācijas funkcijas slāņa pielietošanas, tad $f_{i,j}^l = \sigma \circ z_{i,j}^{l-1}$.

Kā arī šajā vienādojumā ar $Y_{i,j}^{l-1}(r,s)$ ir pierakstīts fragments no slāņa ieejas tenzora $Y_j^{l-1}(r',s'), \quad (r',s') \in \{r - h_1, s + h_1\} \times \{r - h_2, s + h_2\}$. Tātad katrai konvolūcijai/konvolūcijas kodolam atbilst viens neirons.

Redzam, ka lai tālāk pārrakstītu konvolūciju slāni neironu formā, nepieciešams viena neirona summācijas funkciju $z^l_{i,j}$ pielietot vairākas reizes dažādām ieejas vērtībām viena slāņa ietvaros atkarība no r un s. Šo aspektu neiekļauj vienvirziena neironu tīkla definīcija iepriekšējā nodaļā.

Izrakstot konvolūciju summācijas funkcijas formā, ir iespējams konvolūciju slāni pierakstīt kā $Y^l_i(r,s) = \sigma(s^l_i(y^{l-1}_{r,s})) = n^l_i(y^{l-1}_{r,s})$.

2.5. Attēla lokālo īpašību hierarhija

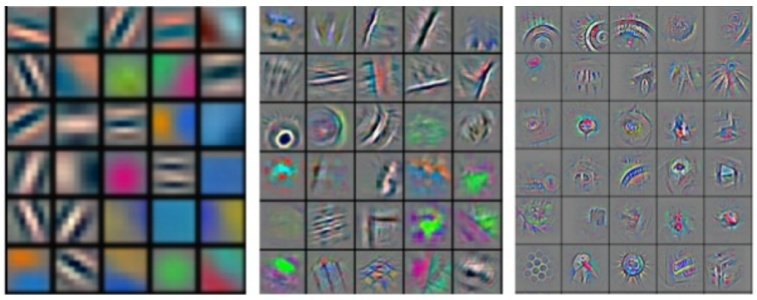
Tiek apskatīts gadījums, kur tīkla pirmais slānis ir konvolūciju slānis. Kā jau tika minēts, konvolūcijas darbojas kā indikators, tam cik ļoti attēla fragments dotam r un s ir līdzīgs konvolūcijas kodolam$K^l_{j,l}$, kurš raksturo attēla lokālu īpašību.

Nākamie slāņi tiek veidoti tā lai tīkls apkopotu informāciju no iepriekšējā slāņa, tādā veidā 'būvējot īpašību hierarhiju'. Tas var tikt veikts vairākos veidos.

- Apvienošanas slānis (*pooling layer*) - ieejas tenzora vērtības tiek agrigētas pa platuma un augstuma dimensijām tās sadalot mazākās daļās, tādā veidā samazinot ieejas tenzora rezolūciju pa platuma un augstuma dimensijām. Par agregācijas funkcija parasti izvēlas maksimuma (*MaxPool*) vai vidējās vērtības (*AvgPool*) funkcionāli.

- Konvolūcijas slānis - konvolūciju kodols tiek "slidināts" izlaižot 1 vai vairākus indeksus, pa augstuma, platuma vai dziļuma dimensijām.

Tādā veidā tīkls ir spiests iemācīties izveidot īpašību hierarhiju.



2.2. att. Sākot no kreisās puses, var redzēt, kā formas un krāsas kļūst sarežģītākas

Attēli 2.2 ir iegūti aprēķinot tīkla ieejas vērtības pie dotām slāņu ieeju vērtībām (inversējot neironu tīkla apakšgrafu funkcijas F^j, apzīmējums no nodaļas par neironu tīkliem).

Interesanti, ka empīriskos novērojumos, apmācot dažādu arhitektūru konvolūciju tīklus, tika novērots, ka platuma un augstuma dimensiju samazināšana un dziļuma dimensijas palielināšana liek tīklam labi iemācīties īpašību hierarhiju, kur augstākas īpašības sastāv no vairākām vienkāršām īpašībām. Un lai veiksmīgi iemācītos daudzas zemo īpašību kombinācijas, augstāko īpašību skaits tiek palielināts.

2.6. Atpakaļatgriezeniskais algoritms konvolūciju tīklam

Lai funkcija $n_{i,j}^l : X_{i,j}^l \to Y_{i,j}^l$ ir neirons ar pēcteču kopu $J_{i,j}^l(r,s)$. Svarīgi ievērot, ka pēcteču kopa ir atkarīga arī no r un s, tas tāpēc, ka šis neirons tiek pielietos vairākām ieejas vērtībām viena slāņa ietvaros. Tad parciālatvasinājuma pēc parametra $w_{u,v}^{i,j,l}$ izteiksme ir

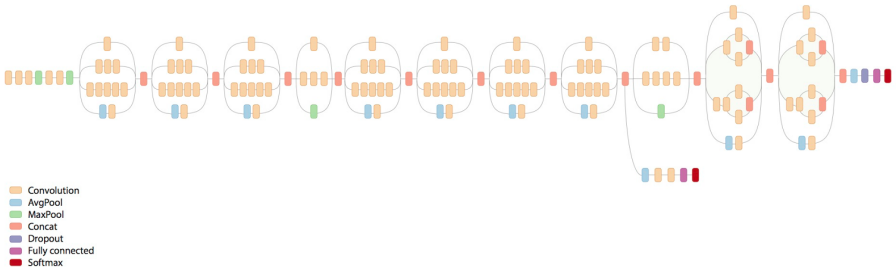
$$\frac{\partial F}{\partial w_{u,v}^{i,j,l}}(x_{i,j}^l) = \sum_{r,s} \frac{\partial f_{i,j}^l}{\partial w_{u,v}^{i,j,l}}(x_{i,j}^l) \left(\sum_{(a,b)\in J_{i,j}^l(r,s)} \frac{\partial f_{a,b}^{l+1}}{\partial y_{i,j}^l}(x_{a,b}^{l+1}) \right),$$

līdz ar to tik un tā var pielietot atpakaļatgriezensikā algoritmu, tikai ar nelielām modifikācijām.

Konvolūciju tīkli parasti ir vairākus slāņus dziļi un arī cieš no sarūkošā un eksplodējošā gradienta problēmām, tāpēc praksē gandrīz vienmēr konvolūciju slāņu aktivizācijas funkcijas ir RELU vai RELU6.

2.7. Inception V3 konvolūciju tīkls

Attēlu aprakstīšanas modelī tiek izmantots *Inception V3* 2.3 konvolūciju modelis. Šis konvolūciju tīkls ir izstrādāts lai atpazītu objektus attēlos.



2.3. att. *Inception V3* konvolūciju tīkls objektu atpazīšanai

Tam ir vairāk kā 20 miljoni parametri un ir vairākus desmitus slāņus dziļš.

Attēlu aprakstīšanas modelī tika iekļauts šis konvolūciju tīkl, jo 2015. gadā tas bija modelis ar augstāko precizitāti attēlu atpazīšanā [13].

3 Rekurentie neironu tīkli

Rekurentie neironu tīkli ir populāri modeļi laikrindu modeļi, kuri pēdējos gados ir izrādījušies ļoti noderīgi daudzos naturālās valodas apstrādāšanas uzdevumos. Šis īsais rekurento neironu tīklu ievads balstīts uz mācību materiāla [5].

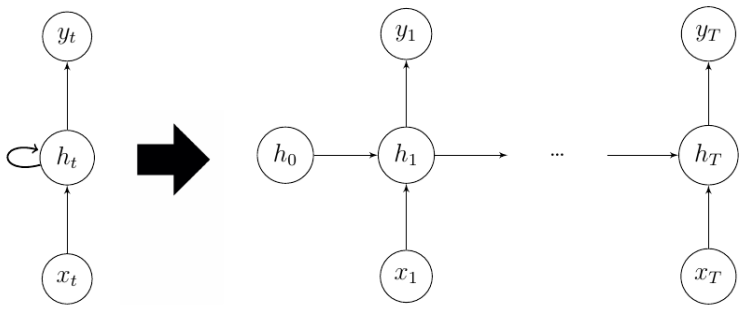
Vienvirziena tīkli pieņem, ka ieejas vektori, kas tiek padoti tīklam ir neatkarīgi, tīkls nesaglabā nekādu informāciju par pagātnē novērotajām ieejas vērtībām. Daudziem uzdevumiem šis arhitektūru tips ir ļoti atbilstošs, bet citos tas ir uzskatāms par lielu ierobežojumu. Piemēram, ja uzdevums ir prognozēt nākamo vārdu teikumā, svarīgi zināt kādi vārdi ir bijuši pirms tā, kā arī stāstījuma konteksts.

RNN var arī saukt par ilgtermiņa atmiņas laikrindu modeļi, tie var saglabāt informāciju patvaļīgus laika soļus uz priekšu. Taču praksē RNN ir grūti apmācīt izprast daudzus laika soļus garas atkarības.

Teorētiski galvenā atšķirība, kas atšķir vienvirziena tīklus no RNN ir tas, ka RNN grafos ir atļauti cikli. Tieši šo ciklu dēļ tīkls spēj padot sev nākamajos laika soļos kaut kādu novērotu informāciju no pagātnes un ir piemērots virkņu datu - laikrindu modelēšanai. Ar (x_t) sapratīsim modeļa ieejas vektoru virkni, ar (h_t) - tīkla slēpto stāvokļu kopu un (y_t) - izejas vektoru virkni.

Ar RNN modeli parasti saprot attēlojumu $(h_{t-1},x_t) \to h_t$, dažreiz RNN tiek attiecināts arī uz izejas virknes vērtību aprēķināšanu $(h_{t-1},x_t) \to (h_t, y_t)$. Šajā nodaļā ar RNN tiks saprasts $(h_{t-1},x_t) \to h_t$ gadījums.

Tātad tīkla slēptie stāvokļi jeb neironu slāņi, kuru virsotnes atrodas ciklā, saglabā informāciju par to kas tikts "parādīts" tīklam iepriekšējos laika soļos. Pateicoties šim arhitektūras elementam, RNN spēj "atcerēties", kas tiem parādīts patvaļīgus laika soļus atpakaļ.



3.1. att. Shēmā attēlots kā RNN "atritināta" pa laika soļiem (virsotnes reprezentēt slāņus)

Bieži RNN tiek apskatīti "atritinātā" veidā 3.1. Nozīmīgs punkts ir tas, ka atritinātais RNN ir tas pats, kas vienvirziena tīkls ar vienādiem parametriem katrā laika solī. Šīs līdzības dēļ populārākais RNN apmācības algoritms ir atpakaļatgriezeniskā algoritma veids - atpakaļatgriezeniskais algoritms laikā (*backpropogation through time*), kas savā vienkāršākajā formā darbojas tieši tāpat kā parastais BP ar atšķirību, ka starp laika soļiem parametri ir nemainīgi.

Šajā nodaļā neironi netiks apskatīti sīkāk kā tikai slāņu līmenī vienkāršības labad.

3.1. Vienkāršie rekurentie neironu tīkli

Vienkāršos rekurentos neironu tīklus (SRNN) definē ar vienādojumu

$$h_t \;=\; \sigma(Wh_{t-1} + Ux_t + b),$$

kur W,U, b ir tīkla parametru masīvi, $t = 1,...,T$ un h_0 var būt novērtēts kopā ar parametriem, saturēt tikai nulles, vai kalpot kā modeļa ieejas arguments. Izejas vērtības no slēptajiem stāvokļiem var būt atkarīgi dažādos veidos. Izejas virknes vērtības $(y_t)_{1,...,T}$ var būt atkarīgas no (h_t) daudzos veidos, vispopulārākie ir

- $y_t = g(h_t) = \sigma(Vh_t)$, katram $t \in \{1,...,T\}$, šādu sakarību izmanto lai piekārtotu virkni (y_t) virknei (x_t),
- virkne sastāv tikai no viena elementa $y_T = \sigma(Vh_T)$, šādu sakarību izmanto lai, piemēram, klasificētu virknes (x_t), tad $y_T \in \{0,1\}$.

3.2. SRNN kā Tjūringa mašīna

Kā interesantu faktu var minēt ka ar patvaļīgas arhitektūras SRNN var aprakstīt jebkuru Tjūringa mašīnu. Darbā [6] konstruēja SRNN ar 1058 slēptā slāņa neironiem (ar racionāliem, fiksētiem koeficientiem), kurš spēja imitēt universālo Tjūringa mašīnu. Tika pierādīts arī otrādāk - ka Tjūringa mašīna spēj imitēt tikko pieminēto RNN ar racionāliem parametriem.

Tātad SRNN modeļu ekspresivitāte ir milzīga - tie spēj aprakstīt jebkuru algoritmu - jebkuru programmu. Problēma ir tajā, ka SRNN apmācība nav tik vienkārša, iemācīties "sarežģītas" programmu tikai no tās ieejas un izeju piemēriem ir, pagaidām, neiespējami.

Šīs nodaļas beigās tiks apskatīts piemērs, kur ar SRNN iemācīsies vienkāršu galīgu automātu, kurš spēj atpazīt regulāru valodu.

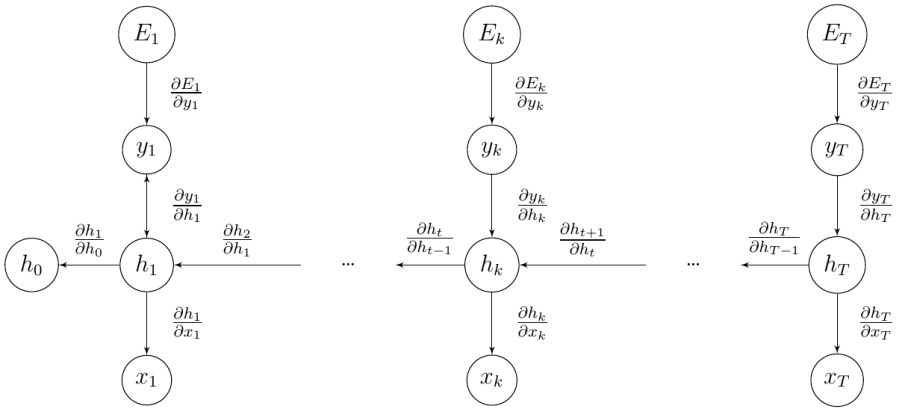
3.3. SRNN apmācība

Kā jau iepriekš tika minēts, SRNN apmācībai izmanto BPTT algoritmu. Šis algoritms izmanto SRNN "atritinātajā" veidā 3.1. Vienkāršības labad tiks apskatīta kļūdas funkcija

$$E = \frac{1}{n}\sum_{i=1}^{n} E^i$$

vienai datu virknei $x := (x_t)$, kur katrā laika momentā t virknes elements x_t ir vektors. Tad kļūdas funkcija pieņem formu

$$E = E^1 = \sum_{t=1}^{T} E_t^1 = \sum_{t=1}^{T} (l_t^1 - y_t^1)^2.$$



3.2. att. Gradienta apreķināšanas shēma atritinātam RNN

Atritinātam RNN kļūdas tiek rēķinātas katrā laika solī atsevišķi, tad saskaitītas kopā. Parciālos atvasinājumus var iegūt Jakobiānus reizinot, ja tie atrodas virknē, un saskaitot rezultātus, kur divi ceļi savienojas, kā tas attēlots shēmā 3.2. Tagad atvasināšana tiks aprakstīta konkrētāk balstoties uz pētījumu [7]. Tiks sākts ar E_T atvasinājumu pēc viena no matricas W parametra w_i^j.

$$\begin{aligned}
\frac{\partial E_T}{\partial w_i^j} &= 2(y_T - l_T)\frac{\partial g}{\partial s_T}\frac{\partial h_T}{\partial w_i^j} \\
&= 2(y_T - l_T)\frac{\partial g}{\partial h_T}\left(\sum_{t=1}^{T}\frac{\partial h_T}{\partial h_t}\frac{\partial^+ h_t}{\partial w_i^j}\right) \\
\frac{\partial h_T}{\partial h_t} &= \prod_{k=T}^{t+1}\frac{\partial h_k}{\partial h_{k-1}} = \prod_{k=T}^{t+1} W^T diag\left(\sigma'(h_{k-1})\right),
\end{aligned}$$

kur $\frac{\partial^+ h_t}{\partial w_i^j}$ apzīmē parciālo atvasinājumu, kur tiek izmantotas h_{t-1} vērtības nevis kā funkcija, un $\frac{\partial h_T}{\partial h_t}$ ir h_t atvasinājuma matrica pēc h_{t-1}, jeb Jakobiāns.

Tātad parciālais atvasinājums $\frac{\partial E_T}{\partial w_i^j}$ ir arī summa, kuru elementi $\frac{\partial h_T}{\partial h_t}\frac{\partial^+ h_t}{\partial w_i^j}$ raksturo kā atvasinājumu ietekmē tieši t-tais solis, šie elementi tiks saukti par **laika soļu ietekmēm**. Turpmāk tiks atšķirtas ilglaicīgās un īslaicīgās laika soļu ietekmes, kur ilglaicīgajām $t << T$.

3.4. Eksplodējošie un pazūdošie gradienti RNN kontekstā

Tā kā atritināts SRNN ir dziļais neironu tīkls, tad arī šī arhitektūra cieš no pazūdošā un eksplodējošā gradienta problēmas. Šīs parādības saistāmas ar ilglaicīgo laika soļu ietekmju vērtību strauju pieaugumu vai samazināšanos, kuras var augt eksponenciāli pret $T - t$. Šajā paragrāfā aprakstītais šīs problēmas cēlonis ir arī ņemts no [7].

Lai labāk izprastu šo situāciju, tiks apskatītas ilglaicīgās laika soļu ietekmes, kas ir $2(T - t)$ matricu reizinājums. Ar normas operatoru $\|\cdot\|$ kvadrātiskām matricām tiks apzīmēta lineāro operatoru norma un ar indeksiem $j \in \{1,...,dim(h_t)\}$ tiks indeksēti neironi, kuri veido attēlojumu $(h_{t_1},x_t) \to h_t$ slāni.

Pieņemsim, ka $|\sigma'(h_t)_j|$ ir ierobežots, tad $\|diag(\sigma'(h_t))\| \leq \gamma \in R$ balstoties uz to, ka matricas īpašvērtības ir $\sigma'(h_t)$.

Pieņemsim, ka λ_1 ir matricas W lielākā īpašvērtība un $\lambda_1 < \frac{1}{\gamma}$. Lineāra operatora $W^T diag(\sigma'(h_t))$ norma ir ierobežota ar W^T un $diag(\sigma'(h_t))$ normām (Košī-Švarca nevienādība), tad

$$\forall k, \|\frac{\partial h_T}{\partial h_t}\| \leq \|W^T\| \cdot \|diag(\sigma'(h_t))\| < \gamma\frac{1}{\gamma} < 1.$$

Tad eksistē tāds $\eta \in R$, ka $\forall k, \|\frac{\partial h_T}{\partial h_t}\| \leq \eta < 1$, tad mēs iegūstam, ka

$$\|2(y_T - l_T)\frac{\partial g}{\partial h_T}\left(\sum_{t=1}^{T}\frac{\partial h_T}{\partial h_t}\frac{\partial^+ h_t}{\partial w_i^j}\right)\| \leq \eta^{T-t}\|2(y_T - l_T)\frac{\partial g}{\partial h_T}\|.$$

Līdz ar to lieliem $T - t$ laika komponente tiecas uz nulli un garas atkarības laikā modelis neiemācās.

Analogi manifestējās eksplodējošā gradienta problēma, kad $\lambda > \frac{1}{\gamma}$. Konstantes γ vērtība ir atkarīga no aktivizācijas funkcijām - sigmoīda funkcija $\gamma = 1/4$, hiperboliskā tangensa funkcijai $\gamma = 1$.

Lai cīnītos ar **eksplodējošā gradienta** problēmu, dažādos veidos tiek kontrolēti gradienta komponenšu lielumi.

- *Gradientu ilglaicīgo laika soļu ietekmju apgriešana* - kad laika soļu ietekmes tiek ierobežotas ar kaut kādu konstanti c no augšas, ja laika soļu ietekme pārsniedz soli, tad tās vērtība tiek aizstāta ar c.

- *Gradientu normēšana (gradient clipping)* - kad visa gradienta $\frac{\partial E}{\partial W}$ norma pārsniedz kaut kādu konstanti c, tad gradients tiek aizstāts ar vektoru, kura norma ir vienāda ar c,

$$\frac{c}{\|\frac{\partial E}{\partial W}\|}\frac{\partial E}{\partial W}.$$

Un, lai risinātu **pazūdošā gradienta** problēmu, var pielietot šādas metodes:

- pielāgot RNN arhitektūru, piemēram, LSTM vai GRU.

3.5. LSTM

Lai cīnītos ar pazūdošā gradienta problēmu publikācijā [8] tika ieteikts izmantot *īslaicīgās-ilglaicīgās atmiņas (LSTM)* arhitektūru. Lai gan arī agrāk tika izmantota "vārtu" tehnika rekursīvajos neironu tīklos arī iepriekš, ar LSTM vārtu tehniku sāka izmantot biežāk dažādās RNN konstrukcijās.

Praksē tiek plaši izmantota paplašināta LSTM arhitektūra - papildināta ar aizmiršanas vārtiem, aprakstīta darbā **??**. Tiks sākts tieši ar paplašināto arhitektūru. LSTM raksturo sekojoši vienādojumi

$$\begin{aligned}
input_t &= W_{in}h_{t-1} + U_{in}x_t, \\
ingate_t &= W_{ing}h_{t-1} + U_{ing}x_t, \\
fgate_t &= W_f h_{t-1} + U_f x_t + b, \\
outgate_t &= W_{out}h_{t-1} + U_{out}x_t, \\
c_t &= c_{t-1} \otimes \sigma(fgate_t) + \sigma(input_t) \otimes \tanh(ingate_t), \\
h_t &= \tanh(c_t) \otimes \sigma(outgate_t),
\end{aligned}$$

kur ar $\otimes$ ir apzīmēta vektoru reizināšana pa komponentēm. Arhitektūras komponentes tiek aprakstītas šādi:

- $input_t$ - ieejas vērtību apstrāde;

- $ingate_t$ - ieejas vārti, regulē kas no apstrādātajām ieejas vērtībām tiks saglabāts atmiņas sūnā c_t;

- $fgate_t$ - aizmiršanas vārti, regulē atmiņas šūnu vērtības;
- $outgate_t$ - regulē kas no atmiņas šūnas tiek saglabāts slēptajā stāvoklī h_t;
- c_t - saglabā informāciju no iepriekšējiem laika soļiem, spēj saglabāt informāciju ļoti daudzus laika soļus, tiek saukts par **atmiņas šūnām**;
- h_t - arī saglabā informāciju no iepriekšējiem laika soļiem, taču nevar saglabāt daudzus laika soļus, tiek saukts par **slēpto stāvokli**.

Apskatam ilglaicīgās laika soļu ietekmes LSTM atmiņas šūnām, kur ar c_t^j apzīmējam atmiņas šūnas j-to komponenti ja $j \neq k$:

$$\begin{aligned}
\frac{\partial c_t^j}{\partial c_{t-1}^k} &= \frac{\partial c_{t-1}^j}{\partial c_{t-1}^k}\sigma(fgate_t)_j + \frac{\partial \sigma(fgate_t)_j}{\partial c_{t-1}^k}c_{t-1}^j + \\
&+ \tanh(ingate_t)_j\frac{\partial \sigma(input_t)_j}{\partial c_{t-1}^k} + \sigma(input_t)_j\frac{\partial \tanh(ingate_t)_j}{\partial c_{t-1}^k} = \\
&= 0 + \sigma'(fgate_t)_j w_f^{jk} c_{t-1}^j + \\
&+ \tanh(ingate_t)_j\sigma'(input_t)_j w_{in}^{jk} + \sigma(input_t)_j \tanh'(ingate_t)_j w_j^{jk},
\end{aligned}$$

un ja $j = k$, tad

$$\begin{aligned}
\frac{\partial c_t^j}{\partial c_{t-1}^j} &= \sigma(fgate_t)_j + \sigma'(fgate_t)_j w_f^{jj} c_{t-1}^j + \\
&+ \tanh(ingate_t)_j\sigma'(input_t)_j w_{in}^{jj} + \sigma(input_t)_j \tanh'(ingate_t)_j w_{ing}^{jj}.
\end{aligned}$$

Lai vieglāk redzētu, kas notiek šajos vienādojumos pārrakstam šos vienādojumus matricu formā (jo $\frac{\partial c_t}{\partial c_{t-1}}$ ir Jakobiāns)

$$\begin{aligned}
\frac{\partial c_t}{\partial c_{t-1}} &= diag(\sigma(fgate_t)) \\
&+ W_f^T diag(\sigma'(fgate_t)c_{t-1}) \\
&+ W_{in}^T diag\left(\sigma'(input_t)\tanh(ingate_t)\right) \\
&+ W_{ing}^T diag\left(\sigma(input_t)\tanh'(ingate_t)\right).
\end{aligned}$$

Redzam, ka Jakobiāns ir atkarīgs no vairākiem summas locekļiem, kuri var svārstīties pietiekami neatkarīgi viens no otra lai $\frac{\partial c_k}{\partial c_{k-1}}$ norma svārstītos ap 1, līdz ar to $\| \prod_{k=t}^{T} \frac{\partial c_k}{\partial c_{k-1}} \|$

norma eksponenciāli nepieaugt vai netiekties uz nulli. Kas nodrošina ilglaicīgo laika soļu "stabilitāti".

Tā kā c_t jau satur "ilglaicīgo" informāciju, tad nav svarīgi lai h_t arī to saturētu, līdz ar to h_t izteiksme parasti nav pielāgota ta lai izvairītos no pazūdošā gradienta problēmas.

Tā kā oriģinālajā LSTM arhitektūrā nebija aizmiršanas vārtu, tad atmiņas šūnas vienādojums oriģinālajam modelim ir šāds

$$c_t = c_{t-1} + \sigma(input_t) \otimes \tanh(ingate_t).$$

Ideja šeit ir līdzīga kā dziļajos NN izmantojot RELU aktivizācijas funkciju - lai Jakobiāni $\frac{\partial c_t}{\partial c_{t-1}} \sim I$, un gadījumos, kad $input_t$ un $ingate_t$ nav atkarīgi no c_{t-1}, tad $\frac{\partial c_t}{\partial c_{t-1}} = I$, kur I ir vienības matrica.

Aizmiršanas vārtus sāka lietot sekojošu iemeslu dēļ.

- Dažādām parādībām laikā, piemēram, kontekstiem teikumā, var būt dažāds garums, līdz ar to aizmiršanas vārti atgriež šūnas sākotnējā stāvoklī, ja, piemēram, dotā ieeja signalizē, par parādības beigām, piemēram, ar punktu beidzas teikuma konteksts, līdz ar to, piemēram, jāizdzēš informācija par teikuma gramatisko centru.
- Publikācijā [9] minēts, ka oriģinālajā LSTM arhitektūra [8], kurai nebija aizmiršanas vārtu, cieta no tā, ka atmiņas šūnu vērtības pieaug neierobežoti atkarībā no t, līdz ar to tās var pārstāt kalpot kā atmiņas šūnas.

3.6. GRU

GRU arhitektūras modeļi arī izmanto vārtu tehniku, lai izvairītos no sarūkošā gradienta problēmas. Šī arhitektūra ieteikta publikācijā [10], kur GRU arhitektūra lietota līdzīgā kontekstā kā attēlu komentēšanā - mašīntulkošanā.

GRU arhitektūru apraksta sekojoši vienādojumi

$$\begin{aligned} r_t &= \sigma\left(W_r h_{t-1} + U_r x_t + b_r\right), \\ z_t &= \sigma\left(W_z h_{t-1} + U_z x_t + b_z\right), \\ \hat{h}_t &= \tanh\left(W_h r_t \otimes h_{t-1} + U_h x_t\right), \\ h_t &= z_t \otimes h_{t-1} + (1 - z_t) \otimes \hat{h}_t, \end{aligned}$$

kur h_t ir slēptais stāvoklis, kas tiek padots nākamajam laika solim. Plašāk tiks aprakstītas GRU komponentes.

- $\hat{h}_t$ sauc par **kandidāt stāvokli**, jo tas satur vērtības, kas potenciāli var tieši aizstāt h_t vērtības, to kā notiks aizstāšana nosaka atjaunināšanas vārti z_t.
- z_t, jeb **atjaunināšanas vārtu**, komponentes nosaka to, cik lielu daļu nākamais GRU stāvoklis saturēs informāciju no kandidāt stāvokļa, līdz ar to h_t darbojas kā adaptīvi izliektas kombinācijas svari.
- r_t nosaka, cik daudz informācijas no h_{t-1} saturēs kandidāt-stāvoklis.

GRU arhitektūra ir vienkāršāka par LSTM. Var redzēt, ka GRU arhitektūra sastāv no mazāk atsevišķām daļām - ir vienkāršāks nekā LSTM arhitektūra, līdz ar to arī vienkāršāk implementējams un apmācāms.

Tā kā slēptais stāvoklis h_t nav tieši saistīts ar h_{t-1}, kā tas ir oriģinālajā LSTM arhitektūrā (bez aizmiršanas vārtiem), tad speciāli aizmiršanas vārti GRU arhitektūrai nav nepieciešami, atjaunināšanas vārti z_t daļēji pilda šo uzdevumu.

GRU ilglaicīgās laika soļu ietekmes necieš no gradienta pazušanas problemātikas tādu pašu iemeslu dēļ kā LSTM.

3.7. DSGU arhitektūra

DSGU (*Deep Simple Gated Unit*) ir vēl viena RNN arhitektūra, kura izmanto vārtu principu lai cīnītos ar pazūdošā gradienta problēmu.

DSGU arhitektūra, kura prezentēta darbā [11], ir vienkāršāka nekā LSTM un GRU, saturot tikai vienus vārtus. Taču ar līdzīgām ekspresivitātes spējām. Kā arī [11] minēts, ka vairākām datu kopām DSGU uzrāda labus rezultātus daudz ātrāk pret apmācības iterācijām nekā GRU vai LSTM.

DSGU arhitektūru definē sekojoši vienādojumi

$$\begin{aligned} x_g &= W_{xh}x_t + b_g, \\ z_g &= \tanh(W_{zxh}(x \otimes h_{t-1})), \\ z_{out} &= \sigma\left(W_{go}(z_g \otimes h_{t-1})\right), \\ z &= \hat{\sigma}\left(W_{xz}x_t + b_z + W_{hz}h_{t-1}\right), \\ h_t &= (1 - z_t) \otimes h_{t-1} + z_t \otimes z_{out}, \end{aligned}$$

kur $\hat{\sigma}(x) = \max(0, \min(1, (x+1)/2))$ sauc par cieto sigmoīda funkciju.

Arhitektūras, kuras izmanto vārtu principu cieš arī mazāk no eksplodējošā gradienta problēmas, taču tas tik un tā notiek, un tiek parasti "ārstēts" ar gradientu apgriešanas metodi.

3.8. Regulāru valodu mācīšanās ar RNN

Šī nodaļa ilustrē spējas iemācīties saglabāt informāciju daudzus laika soļu atpakaļ.

3.8.1. Simulētie dati

Tagad apskatīsim interesantu piemēru, kur ar dažādām RNN arhitektūrām un autoregresīvu modeli tiks akceptētas regulāras valodas

$$([0-9]\{15,20\}[a-z]\{15,20\})^3$$

Lai gan datu kopa ir gadījuma, vārdu kopu no šīs valodas un vārdu kopu ne no šīs valodas ir strikti atšķiramas. Līdz ar to perfektajā gadījumā neironu tīklam vajadzētu atpazīt ar 100% precizitāti.

Regulārā valoda, kuras vārdi tika simulēti, ir relatīvi vienkārša, taču, lai to sekmīgi iemācītos atpazīt, tīklam ir jāiemācās saglabāt, kurš no etapiem $[0-9]\{15,20\}[a-z]\{15,20\}$ ir pagājis jeb "jāiemācās skaitīt". Kā arī atmiņai vajadzēs stiepties līdz apmēram 100 laika soļus, līdz ar to varēs redzēsim arī, kā rekurento neironu tīklu arhitektūras, kuras neizmanto vārtu tehniku, nespēs vai ļoti lēni mācīsies šo valodu.

Valodas alfabēts satur 46 simbolus - ciparus un angļu valodas alfabēta burtus.

3.8.2. Izmantotās Neironu tīklu arhitektūras

Šim ”rotaļu” piemēram tika izmantotas šādas rekurento tīklu arhitektūras - SRNN, LSTM, GRU un DSGU.

Visiem tīkliem slēpto slāņu dimensija ir 6, LSTM gadījumā gan atmiņas šūna, gan slēptais stāvoklis ir ar dimensiju 6. Tikai pēdējais rekurento tīklu slānis h_T tika izmantots, lai veidotu izejas vērtību $y = \sigma(Wh_T)$. Šī ir tipiska rīcība virkņu klasificēšanas gadījumos. Kā arī ieejas $x_t \in \mathbb{R}^{46}$ tika pārveidotas ar lineāru transformāciju uz $\mathbb{R}^6$, kas bija nepieciešams visām arhitektūrām ar vārtiem.

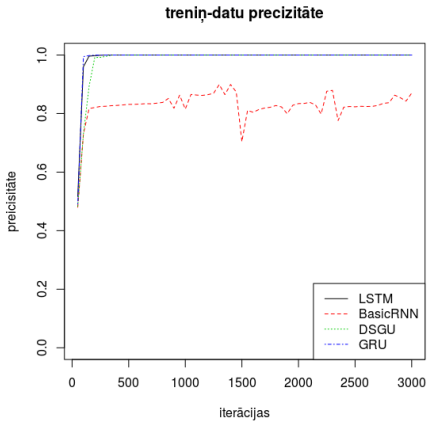
Jau uzreiz skaidrs, ka auteregresīviem laikrindu modeļiem

$$x_t = f(x_{t-1},...,x_{t-d}) + \epsilon_t$$

tas nebūtu pa spēkam, jo to ”atmiņa” ir ierobežota d soļus. Tāpēc arī valodas modelēšanai daudz piemērotāki ir ilglaicīgās atmiņas modeļi kā markova slēpti modeļi un rekurentie neironu tīkli, jo vārdu ietekme var stiepties vairākus vārdus vai teikumus uz priekšu. Tāpēc šie modeļi netika izmantoti šajā ilustrācijā, kā arī to implementācija būtu sarežģīta šajā kontekstā.

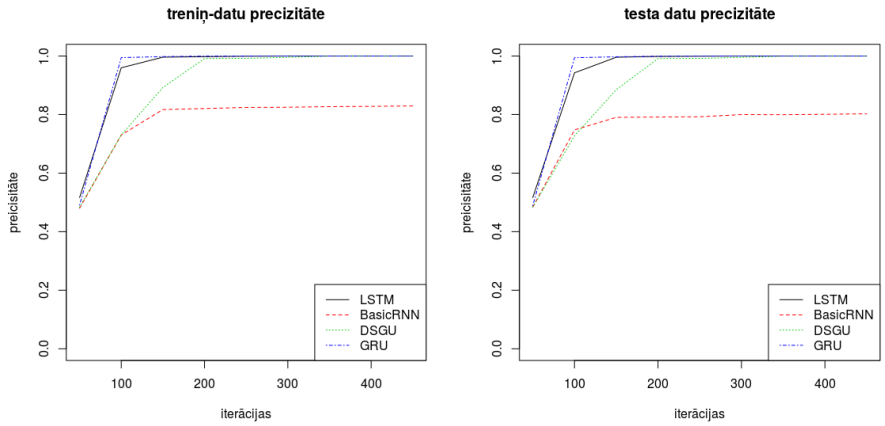
3.8.3. Apmācīšanas rezultāti

Katrai neironu tīklu arhitektūrai tika izpildītas 3000 atpakaļatgriezeniskā algoritma iterācijas, ja precizitāte pārsniedz 0.999 apmācības datu kopai, tad apmācība tika pārtraukta.



3.3. att. Modeļu precizitāte apmācības datu kopai atkarībā no algoritma iterācijām

Grafikā 3.3 var redzēt modeļu sniegumu. Kā jau bija gaidīts - parastā RNN arhitektūra nespēja iemācīties daudzu laika soļu garās atkarības. Taču pārējās arhitektūras pārsniedza 0.999 precizitāti mazāk kā 500 iterācijās.



(a) Apmācības datu kopai (b) Testa datu kopai

3.4. att. Modeļu precizitāte atkarībā no algoritma iterācijām līdz 500 iterācijām

Grafikā 3.4 var novērot, ka GRU ar LSTM iemācījās atpazīt valodu apmēram vienādi ātri, DSGU atpalika.

Ilustrācijas programmu var apskatīt autora *GitHub* repozitorijā [19].

4 Reprezentāciju attēlojumu mācīšanās

Attēlu aprakstīšanas neironu tīkls izmanto reprezentāciju mācīšanās jēdzienu, tāpēc šajā nodaļā tiks īsi apskatīts šis plašais lauks. Šis apraksts balstīts uz materiālu [**?**].

Daudzos mašīnmācīšanās pielietojumos ieeju telpa $X \subseteq \mathbb{R}^d$ ir ar ļoti augstu dimensiju. Piemēram, ieejas telpa ir vārdi no vārdnīcas ar izmēru 12000, vai attēls - 300×300 dimensijas. Augstais dimensiju skaits palielina nepieciešamo resursu apjomu neironu tīkla apmācībai, kā arī visdrīzāk satur daudz nevajadzīgas informācijas - troksni.

Parasti datu punkti no daudz-dimensionālas telpas X pieder apakškopai $R \subset X$ ar daudz mazāku dimensiju skaitu $dim(R) << dim(X)$.

R tiks saukta par reprezentāciju telpu, un $r : X \to R$ par reprezentāciju attēlojumu.

Reprezentāciju jēdziens nav saistīts tikai ar dimensiju samazināšanu, bet arī ar noderīgo īpašību (*features*) saglabāšanu un atšķiršanu no nevajadzīgajām īpašībām (atkarīgām no uzdevuma kontekstam) vai trokšņa.

Piemēram, kompjūter-redzes kontekstā objektu klasificēšanā objektiem var būt īpašība r_1, kura ir atkarīga no gaismas avota pozīcijas attiecībā pret objektu, un īpašība r_2, kura nav no tā atkarīga. Šajā gadījumā laba objekta reprezentācija objektu atpazīšanas kontekstā saturētu r_1 nevis r_2.

Parasti reprezentācijas iedalās divās lielās grupās. Šīs grupas tiks ilustrētas ar vienkāršu piemēru. Lai A ir objektu kopa - $A = \{a_1,...,a_4\}$ = {apelsīns, ābols, bumbieris, banāns}. Tad ir divi fundamentāli atšķirīgi veidi kā reprezentēt šos objektus. Ar **atmiņas sistēmu** tiks saprasts skaitļu vektors $(x_1,....,x_n) \in X \subseteq \mathbb{R}^n$, vektora komponentes tiks sauktas par atmiņas šūnām. Šajā augļu piemērā tiks pieņemts, ka $x_i \in \{0; 1\}$.

- Par **lokālām reprezentācijām** sauc atmiņas sistēmas, kur katrs objekts tiek reprezentēts ar vienu atmiņas šūnu 4.1.

4.1. tabula. Augļu lokāla reprezentācija

a_i	$r(a_i)$
apelsīns	1000
ābols	0100
bumbieris	0010
banāns	0001

4.2. tabula. Augļu sadalīta reprezentācija

a_i	$r(a_i)$
apelsīns	10
ābols	01
bumbieris	11
banāns	00

- **Sadalītās reprezentācijas** (*distributed representation*) katram objektam piekārto vairāku atmiņas šūnu kombināciju 4.2.

Abiem reprezentāciju viediem ir savi plusi un mīnusi 4.3.

Ar neironu tīkliem bieži tieši vai netieši tiek meklētas labas objektu sadalītās reprezentācijas. Parasti tiek dota objektu lokālā reprezentācija un meklēts tiek reprezentāciju attēlojums $r : X \to Y$, kurš reprezentē lokālo reprezentāciju par sadalīto.

Labu sadalīto reprezentāciju raksturīpašības ir:

- kā jau iepriekš minēts - telpai Y ir daudz mazāka dimensija nekā X;
- ja x_1,$x_2 \in X$ ir "tuvi" pēc kaut kāda distances mēra, tad arī $y_1 = r(x_1)$,$y_2 = r(x_2) \in Y$ ir "tuvi";
- ja starp objektiem x_1,x_2,x_3,$x_4 \in X$ pastāv kaut kāda semantiska relācija $x_1 \xrightarrow{l} x_2$, tad reprezentāciju telpā Y objektu reprezentācijas y_1 un y_2 ir tādas, ka zinot tikai divus objektus no trijnieka (y_1,l,y_2), mēs var atrast trešo, izmantojot kaut kādas vienkāršas aritmētiskas darbības.

4.3. tabula. Reprezetnāciju priekšrocības

sadalītā	lokālā
Nepieciešams mazāks atmiņas vienību skaits.	Vieglāk papildināma, maināma.
Papildus pašiem objektiem, var reprezentēt attiecības/relācijas starp objektiem.	
Parasti sagrupē objektus ar līdzīgām īpašībām.	

Pēdējā no vēlamajām īpašībām ir visgrūtāk izpildāmā un visinteresantākā. Relāciju jēdziens būs ļoti nozīmīgs attēlu aprakstīšanas modelim.

Lai radītu labāku priekšstatu pat relāciju reprezentāciju, tiks apskatīts vārdu reprezentāciju modelis *Skipgram*.

Lielākā daļa vārdu reprezentāciju apmācību algoritmu balstās uz **vārdu sadalījuma hipotēzi**, kura apgalvo, ka vārdi, kuri bieži tiek parādās kopā vienos tiekumos , dala kaut kādu semantisku nozīmi [23]. *Skipgram* modelis ir balstīts uz šīs hipotēzes.

4.1. *Skipgram* modelis

Skipgram modelis ir neironu tīkls, kurš dotam vārdam $x \in dict$, kur $dict$ ir vārdnīca, piekārto vārdu kopu $\{x_1,...,x_d\} \subset dict$, kuri tiek bieži lietoti kopā ar vārdu x, šo vārdu kopu bieži dēvē par vārda x **kontekstu**. Modeļa apraksts balstīts uz [23].

Par *Skipgram* modeli sauc attēlojumu $F : V \to C$, kur V,$C \subseteq \mathbb{R}^n$ (n ir vārdnīcas izmērs). V sastāv no vektoriem, kur tikai i-tajam vārdam atbilstošā i-tā koordināta ir 1, pārējās koordinātas ir nulles. Un kopa C sastāv no vektoriem, kuri satur varbūtības ka $P(c|a), \quad a \in A \quad \& \quad c \in C$, šīs nosacītās varbūtības ir vārdu c biežumi (dalīti ar biežumu kopsummu) tekstu fragmentos, kuros ir vārds a.

Tīklam ir divi pilnsaistes slāņi ar atbilstošajām matricām W_a un W_a

$$a \to W_a a = y \to softmax(W_c y) \to c.$$

Šis modelis tiek apmācīts maksimizējot varbūtību $\log P(c|a)$.

Apmācot *Skipgram* modeli, vārdu reprezentācijas y tiek iegūtas netiešā veidā, reprezentāciju attēlojums ir $r(a) = W_a a$.

Šīs reprezentācijas satur arī noderīgu semantisku informāciju. Lai salīdzinātu vārdu semantisko nozīmi, ievieš papildus jēdzienu - **kosinusu līdzības mēru**, kas ir funkcija $sim : Y^2 \to R$

$$sim(u,v) = \frac{\sum_{i=1}^{d} u_i v_i}{\sqrt{\sum_{i=1}^{d} u_i^2}\sqrt{\sum_{i=1}^{d} v_i^2}}, \quad d = dim(Y).$$

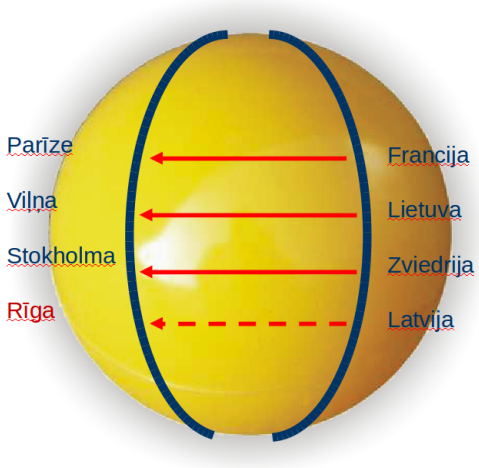
Šis līdzības mērs pieņem tādas pašas vērtības kā kosinusa funkcija no leņķa starp vektoriem u un v, jeb $sim(u,v) = \cos(v \angle u)$. Tātad šis mērs apraksta, vai vektori ir vērsti līdzīgos virzienos, tas neņem vērā vektoru garumus, citiem vārdiem sakot, ja visus vek-

torus attēlotu uz vienības lodes virsmas (vienkārši mainītu vektoru garumus uz 1), tas neizmainītu kosinusa līdzības mēra vērtības starp vektoriem.

Tad reprezentācijas y_1,$y_2 \in Y$ satur semantisko nozīmi jeb bieži parādās līdzīgos kontekstos, ja to kosinusu līdzības mēri ir tuvu 1.

Ļoti interesanta īpašība *Skipgram* reprezentācijām ir tāda, ka ja visi vārdu reprezentāciju garumi tiktu normēti uz 1 jeb reprezentācijas tiktu projicētas uz vienības lodes virsmas S, tad mēs varam uzdot relācijas starp vārdiem kā attēlojumus $l : S \to S$ jeb rotācijas 4.1. Viens no populārākajiem piemēriem ir ar relāciju, kura vārdus "vīrietis" un "sieviete" attēlo par "karalis" un "karaliene". Šo iemācīto relāciju struktūru var izmantot šādi:

- atrod vārdu pāri, kas uzdod relāciju l, piemēram, l("vīrietis") = "karalis";
- atrod rotāciju l no "vīrietis" uz "karalis";
- varam pielietot šo pašu rotāciju vārdam "sieviete", tad vārdam "karaliene" vajadzētu atrasties tuvu l("sieviete").



4.1. att. Relācijas kā rotācijas *Skipgram* reprezentāciju telpā

Reprezentācijas, kuras ir bagātas ar šādām relācijām ir ļoti vērtīgas, jo ļauj mums deducēt līdzības un attiecības - saprast dziļāku teksta semantisko nozīmi.

Šīs reprezentācijas dažreiz var strādāt neprecīzi, jo tās tiek iegūtas neuzraudzītās mācīšanās ceļā, citiem vārdiem sakot, netiek minimizētas šo reprezentāciju kļūdas, bet tās iegūst kā "bonusu" pēc visa *Skipgram* modeļa novērtēšanas. Piemēram, [24] darbā tiek aprakstīta *TransE* relāciju mācīšanās, kas tiek realizēta uzraudzītās mācīšanās manierē.

4.4. tabula. Vārdu reprezentāciju grupas

vārds	vārdi ar augstu kosinusa līdzību
dzimt	mirt, piedzimt, lai arī, pamatskola, nomirt, kopš, domāt, izņemot
gads	g., maijs, gadsimts, jau, kopš, tomēr, mēnesis, lantāns
kāds	kāda, jebkurš, viens, attiecīgs, viss, dažāds, konkrēts, sens
vai	un, jeb, u.c., nevis, nekā, kā arī, utt., bet arī
pa	gar, caur, apkārt, cauri, uz, zem, pakaļ, no
ar	starp, starpa, pateikties, 10,1, pret, krācīte, nonākt, un
šī	šāda, katra, dažāda, abas, daudzas, galvenī, iepriekšēja, tāda
ļoti	diezgan, visai, samērā, tik, salīdzinoši, pietiekami, ārkārtīgi, tikpat
jo	ka, bet, taču, tādēļ, kad, kur, tāpēc, bet arī
sava	savs, mana, tava, viņš, viņa, tavs, pilna, solo

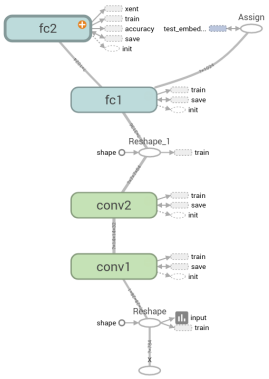
4.1.1. *Skipgram* modelis latviešu valodai

Tika izmantots parauga programmas kods no lekciju materiāliem [25], kas apmācīja *Skipgram* modeli. Tabulā 4.4 var redzēt kā *Skipgram* reprezentāciju attēlojums sagrupēja vārdus grupās. Vārdi, kuri varētu tikt lietoti līdzīgos kontekstos, ir sagrupēti kopā.

4.2. Attēlu reprezentācijas

Attēlu reprezentācijas parasti tiek veidotas no modeļiem objektu atpazīšanai. Pēc attēlu klasifikatoru apmācīšanas, pēdējais slānis, kura izeja veido visa tīkla izeju, tiek noņemts. Tad par reprezentāciju attēlojumu tiek uzskatīts atlikušais tīkls - attēlu klasifikators bez izejas slāņa. Šajā nodaļā tiks apskatītas ar roku rakstīto ciparu reprezentācijas.

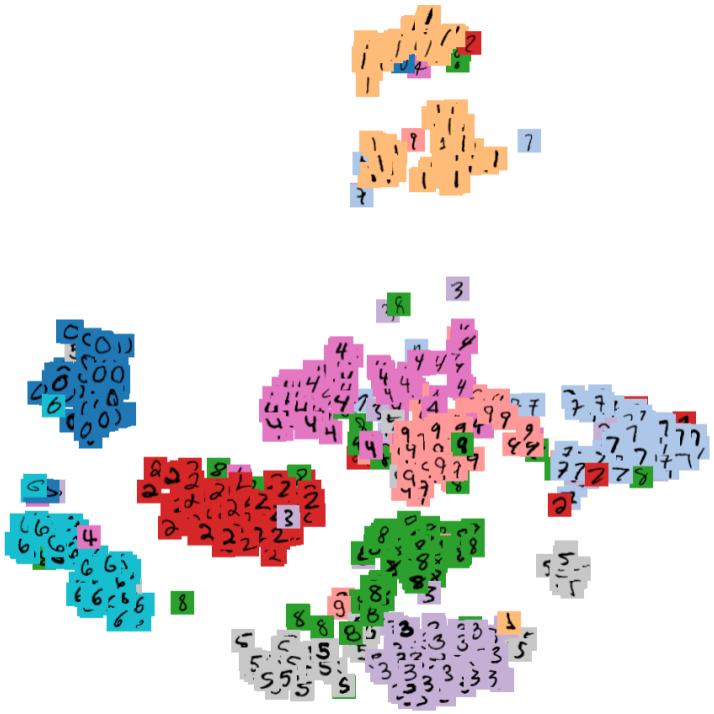
Lai atpazītu ar roku rakstītus ciparus, tika apmācīts neironu tīkls ar diviem konvolūciju slāņiem, kuriem seko divi pilnsaistes slāņi kā attēlots *Tensorflow* grafā 4.2. Attēlu



4.2. att. Neironu tīkls ar roku rakstīto ciparu attēlu klasifikācijai

reprezentācijas tiek iegūtas kā pirmspēdējā pilnsaistes slāņa izejas. Šī reprezentācija ir

vektors ar 1024 elementiem. Lai vizualizētu reprezentācijas, tika izmantota populāra nelineāra metode dimensiju smazināšanai un vizulizēšanai - *t-SNE* [27]. Tikko minētā metode atrod manifoldu ar dimensiju 2 reprezentāciju telpā, lai, projicējot punktus uz šī manifolda, tos varētu attēlot 2-dimensionālā grafikā. Vizualizācijā 4.3 redzams, ka ciparu



4.3. att. Ar roku rakstīto ciparu attēlu reprezentāciju vizualizācija

attēlu reprezentācijas ir sagrupējušās atbilstoši rakstītajam ciparam diezgan precīzi. Kas arī bija gaidāms, jo tīkls tika apmācīts tā, lai nākamā kārta, kura var atdalīt tikai lineāri atdalāmas kopas (grafikā redzams, ka kopas nav lineāri atdalāmas, taču reprezentācijas 1024-dimensionālā telpā ir ļoti tuvu atdalāmām), līdz ar to reprezentāciju attēlojumam jau vajadzēja labi sadalīt ciparu attēlus to klasēs.

Šajā darbā aprakstītais attēlu aprakstīšanas modelis par reprezentāciju attēlojumu izmanto *Inception V3* neironu tīklu bez pēdējā pilnsaistes slāņa. Šīs reprezentācijas ne tikai sagrupē attēlos redzamos objektus, bet satur informāciju par relācijām starp objektiem, kuras tālāk tiek aprakstītas angļu valodas teikumos.

5 Attēlu aprakstīšanas modelis

Attēlu aprakstīšana ir daudz grūtāks uzdevums nekā objektu detektēšana un atpazīšana attēlos. Algoritmam ir ne tikai jāatpazīst objekti, bet arī jāsaprot to relācijas starp tiem, kā arī modelim sevī jāietver valodas ģenerēšanas daļa, kura vizuālo informāciju spēj pārvērst saprotamā, gramatiski pareizā, lasāmā teikumā.

Viens no argumentiem tam, ka relāciju atpazīšana ir sarežģītāka nekā objektu atpazīšana, ir tas, ka uz $O(n)$ objektiem var būt vismaz $O(n^2)$ relāciju, ja starp katriem 2 objektiem ir tikai viena iespējamā relācija, kas parasti tā nav, starp jebkuriem 2 objektiem var būt daudz vairāk relāciju - piemēram, zēns var jāt ar zirgu, stāvēt blakus zirgam, barot zirgu, mazgāt zirgu, utt. Līdz ar to, lai pielietotu relāciju atpazīšanai līdzīgu stratēģiju kā objektu atpazīšanai (kad katram objektam ir vairāki atšķirīgi attēli), tad būtu nepieciešams vismaz kvadrātiski lielāka datu kopa nekā objektu atpazīšanai. Skaidrs, ka relāciju atpazīšanas problemātiku šādi atrisināt nevar. Turklāt teikumi bieži satur relācijas starp vairāk nekā 2 objektiem.

Lai arī šis uzdevums ir grūts, atrisinājums ir ļoti noderīgs, modeli var pielietot, piemēram,

- apvienojot ar runas ģenerēšanas modeli, var tikt radīti dažādi risinājumi neredzīgajiem, piemēram, lai tie spētu saprast bilžu saturu mājaslapās,
- jaunu attēlu datubāžu radīšanai, kur attēli tiks atdoti balstoties uz to aprakstiem,
- šīs problēmas atrisinājums dod noderīgu ieskatu relāciju atpazīšanā.

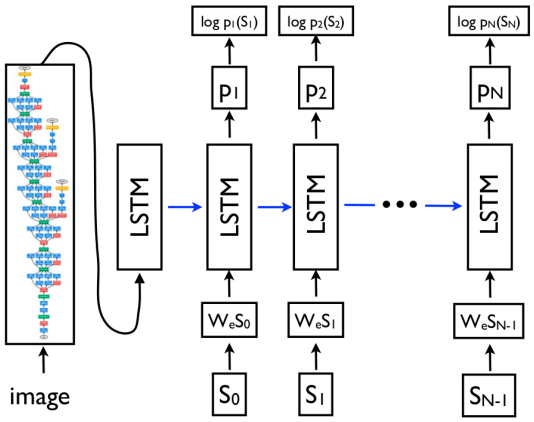
5.1. Modeļa apraksts

Šajā darbā aprakstītais modelis ir izstrādāts, lai piedalītos MSCOCO (*Microsoft Common Objects in Context*) 2015 attēlu komentēšanas sacensībās, kur šis modelis kopvērtējumā

uzvarēja. Šī modeļa autori izstrādāto neironu tīklu ir aprakstījuši publikācijā [14], kā arī *Python* kodu var atrast *GitHub* repozitorijā [15].

Ar i tiks indeksēti attēlu-aprakstu pāri un ar $IV3$ tiks apzīmēts *Inception V3* tīkls bez pēdējā pilnsaistes slāņa. Neironu tīkls attēlu komentēšanai sastāv no divām galvenajā daļām 5.1:

- konvolūciju neironu tīkla *Inception V3*, bez pēdējā pilnsaistes slāņa, attēlu I_i reprezentācijas veidošanai $IV3(I_i) = y_i$;
- LSTM neironu tīkla, kurš ģenerē teikumu angļu valodā $y_i \rightarrow S_i$, kur $S_i = (s_{0,i},s_{1,i},....,s_{n_i,i})$ ir teikums un vārdu lokālās reprezentācijas, no kuriem tas sastāv, un $s_{0,i} =< START >$ un $s_{n_i,i} =< END >$ ir speciāli sākuma un beigu simboli.



5.1. att. Oriģinālā attēlu aprakstīšanas modeļa shēma

Informācijas plūsma modelī notiek šādi:

- attēla I_i reprezentācija $IV3(I_i)$ tiek padota LSTM daļai kā pirmā tīkla atmiņas šūnu vērtība c_0, slēptā stāvokļa h_0 vērtības ir nulles;
- kā ieejas vērtība pirmajā laika solī teikuma ģenerēšanas daļai tiek padots sākuma simbols $< START >$, tālāk katru $t-$to vārdu izvēlas pēc tā, kuram vārdam ir lielākā nosacītā varbūtība

$$P(s_t|I,s_0,...,s_{t-1}) = softmax(W'h_t), \quad 0 <= t < n,$$

kur h_t ir LSTM slēptais stāvoklis;

- kad tiek izvēlēts beigu simbols $< END >$, tad teikuma ģenerēšana tiek beigta.

Turpmāk tikko aprakstītais modelis tiks saukts par oriģinālo modeli.

Šo algoritmu ir iedvesmojušas tulkošanas Neironu Tīklu arhitektūras [10]. Galvenā arhitektūras ideja, lai pārtulkotu teikumu no valodas A uz valodu B, tiek pielietoti divi RNN tīkli.

- Viens RNN pārvērš teikumu par "domu vektorā", jeb šis RNN strādā kā reprezentāciju attēlojums. Šis RNN tiek apmācīts lai "domu vektors" būtu bagāts ar semantisko informāciju.

- Otrs RNN tiek apmācīts lai 'domu vektoru' pārkodētu valodā B.

5.2. Modeļa trennēšana

Modeļa parametri W tiek mainīti tā, lai maksimizētu ticamību attēla aprakstam

$$W^* = \arg\max_W \sum_{(I,S)} \log P(S|I,W),$$

kur varbūtību $P(S|I,W)$ var izrakstīt plašāk

$$P(S|I,W) = \prod_{t=1}^{n} P(S_t|I,S_0,...,S_{t-1},W).$$

Tā kā RNN $h_t = RNN(x_{t-1},h_{t-1})$ ir ilgtermiņa atmiņas modelis, kuram izpildās Markova īpašība, kā arī $p(s_t)$ ir funkcija no h_t, tad var pielietot Markova īpašību vārda s_t nosacītajai varbūtībai

$$P(s_t|I,s_0,...,s_{t-1}) = P(s_0|I) \prod_{l=1}^{t} P(s_l|h_l,s_{l-1}).$$

Apmācības procesā tika izmantots jau apmācīts *Inception V3* modelis, kura parametri apmācības procesā netika mainīti. Darbs koncentrējas uz rekurentā tīkla daļas apmācību. RNN daļa tika apmācīta tam, kā ieejas virkni padod teikumu ar $x_i = (s_{0,i},...,s_{n_i-1,i})$ un kā izejas virkni padodot $y_i = (s_{1,i},...,s_{n_i-1,i})$, tātad ieejas teikumiem trūkst beigu simbols un izejas teikumiem trūkst sākuma simbols.

Kā apmācības algoritms tika izmantots Stohastiskais Gradienta samazināšanās algoritms ar šādām modifikācijām:

- gradientu apgriešanu pie sliekšņa 5;
- sākotnējo apmācības ātrumu (*learning rate*) $\lambda = 2$;
- λ samazināšanās kvocentu 0.5, apmācības ātrums tiek samazināts ik pēc 8 apmācības iterācijām;
- katrā apmācīšanas iterācijā no datiem tika izmantoti 586363 dati.

Visas oriģinālā modeļa variācijas un pats oriģinālais modelis tika apmācīti ar šo apmācības algoritmu.

5.3. Modeļa pielietošana

Modeli pielieto, lai ģenerētu aprakstu/komentāru attēlam, kurš nav apmācības datos. Pieņemsim, ka I ir vel neredzēts attēls un ar $k = 1,...,D$ tiek indeksēti vārdi vārdnīcā. Lai ģenerētu aprakstu attēlam I, tiek veiktas šādas darbības:

- 0-tās atmiņas šūnas vērtības tiek aprēķinātas kā $c_0 := IV3(I)$, LSTM slēptais stāvoklis h_0 ir inicializēts kā nulles;
- pie dotiem c_{t-1},h_{t-1} tiek aprēķināti $(c_t,h_t) = LSTM(c_{t-1},h_{t-1})$;
- tiek aprēķināts varbūtību sadalījums $p_t = softmax(Wc_t)$;
- tiek iegūti 3 vārdi ar augstākajām varbūtībām p_t^k;
- 3 teikumi no $t-1$ soļa tiek papildināti ar t-tajā solī iegūtajiem vārdiem, tad no šīm 9 kombinācijām tiek izvēlēti 3 labākie teikumi pēc to kumulatīvo varbūtību reizinājumiem, šie teikumi tiek uzskatīti par t-tā soļa rezultātu;
- kad kāds teikums tiek papildināts ar beigu simbolu, tad šis teikums tiek uzskatīts par vienu modeļa izejām, tālāk uz soli $t+1$ tiek padoti tikai atlikušie teikumi.

Modeļa pielietošanas algoritms atdod 3 aprakstus par attēlu I.

5.4. Oriģinālā modeļa variācijas

Šī darba ietvaros tika modificēts pieejamais *Python* kods lai eksperimentētu ar jau pieejamo modeļa arhitektūru.

- LSTM tika nomainīts ar GRU arhitektūru. Tā kā mašīntulkošanas uzdevumos GRU arhitektūra labi kalpo kā atkodētājs, tad šai arhitektūrai vajadzētu uzrādīt labus rezultātus arī attēlu aprakstīšanas uzdevumā. GRU arhitektūra oriģināli tika iepazīstināta tieši šādā kontekstā [10].

- Tika implementēta DSGU arhitektūra un pielietota LSTM vietā. Relatīvi labo rezultātu dēļ [11] salīdzinājumā ar LSTM un GRU arī šai arhitektūrai ir potenciāls uzrādīt labus rezultātus šajā uzdevumā.

Tika eksperimentēts ar apmācības algoritmu:

- kā ieejas virkni tika padota $\frac{s_{t,i}+\hat{y}_{t-1,i}}{2}$, kur $\hat{y}_{t-1,i}$ ir modeļa prognozētais vārds $t-1$ solī kā lokāla reprezentācija (D-dimensionāls vektors ar vārda attiecīgo koordināti kā 1, pārējās koordinātas ir nulles), šī apmācības algoritma modifikācija pielietota tikai LSTM arhitektūrai, modeļi, kuri apmācīti ar šādi algoritmu tiks saukti par modeļiem ar **diskrētu atgriezenisko padevi**;

- kā ieejas virkni tika padota $\frac{s_{t,i}+p_{t-1,i}}{2}$, kur $p_{t-1,i}$ ir modeļa $t-1$ laika solī atdotais vārdu varbūtību sadalījums, šī apmācības algoritma modifikācija pielietota visām apskatītajām RNN arhitektūrām, modeļi, kuri apmācīti ar šo algoritma modifikāciju tiks saukti par modeļiem ar **atgriezenisko padevi** .

Šī izmaiņa apmācības procesā tika veikta, lai teikuma S_i ģenerēšanu padarītu līdzīgāku tam, kā tas tiktu ģenerēts jaunam attēlam I_i, līdz ar to, pēc autora domām, varētu palielināt modeļa ģeneralizācijas spējas. Tika eksperimentēts arī padodot tikai $\hat{y}_{t-1,i}$ kā tīkla ieeju t-tajā solī, taču jau pēc dažiem tūkstošiem apmācības iterāciju bija redzams, ka modelim ir lielas grūtības iemācīties loģisku vārdu secību un gramatiku.

5.5. Mašīntulkošanas metrikas

Attēlu aprakstīšanas modeļa izejas vērtība ir idejiski līdzīgas mašīntulkošanas modeļu izejas vērtībām. Līdz attēlu aprakstīšanas modeli var novērtēt ar mašīntulkošanā izstrā-

dātam metodēm.

Vienu teikumu, piemēram, franciski, var iztulkot vairāk kā vienā veidā, tulkojumi var atšķirties ar lietotajiem sinonīmiem, gramatiskajām struktūrām, lai gan tos varētu uzskatīt par vienlīdz kvalitatīviem. Līdz ar to uzdevumus - novērtēt tulkojuma kvalitāti, nav viegli automatizējams. Viskvalitatīvākā tulkojumu kvalitātes novērtēšana ir izmantojot cilvēku - ekspertu, taču tas var būt salīdzinoši (ar automātiskām metodēm) lēni, kā arī var izmaksāt daudz vairāk. Turklāt mašīntulkošanas apjomi ir ļoti lieli cilvēka iejaukšanai.

Mašīntulkošanas kontekstā cilvēka tulkojumus sauc par references teikumiem, un mašīntulkošanas ģenerētos teikumus sauc par kandidāt-teikumiem. Mašīntulkošanas sabiedrībā bieži tiek izmantotas tādas metrikas kā:

- BLEU (*bilingual evaluation understudy*);
- METEOR (*Metric for Evaluation of Translation with Explicit ORdering*);
- ROGUE (*Recall-Oriented Understudy for Gisting Evaluation*).

5.5.1. BLEU metrika

Lai definētu BLEU [16] metriku, vispirms definē modificētu precizitāti (*precision*). Tiek pieņemsim, ka apskata konkrētu tulkojumu (šī darba kontekstā - attēla apraksts), tad, lai $c \in C$ ir kandidāt-teikums no kandidāt-kopas C (lai noteiktu metodes kvalitāti, parasti tiek apskatīti vairāki apraksti vienam attēlam, vai tulkošanas gadījumā - vairāki tulkojumi), un lai $r \in R$ būtu references teikums no references kopas R. Kā arī ar v_n tiks apzīmēta n-gramma. Tad par n-grammu precizitāti sauc

$$p_n = \frac{\sum_{c \in C} \sum_{v_n \in c} \max_{r \in R} \#\{v_n \in r\}}{\sum_{c' \in C} \sum_{v'_n \in c'} \#\{v_n \in c\}},$$

kur $\max_{r \in R} \#\{v_n \in r\}$ ir n-grammas v_n maksimālais skaits kādā no references teikumiem. Tā kā īsiem kandidāt-teikumiem salīdzinājumā ar references teikumiem var būt augstas precizitātes p_n, lai arī tie pilnīgi neatspulguļo referenču saturu, tad tiek lietota konstante BP, kas liek zemākus svarus īsiem kandidāt-teikumiem.

$$BP = \begin{cases} 1 & \text{, ja} \quad \sum_{c \in C} |c| > \sum_{r \in R} |r| \\ \exp(1 - \frac{\sum_{r \in R} |r|}{\sum_{c \in C} |c|}) & \text{,ja savādāk} \end{cases}.$$

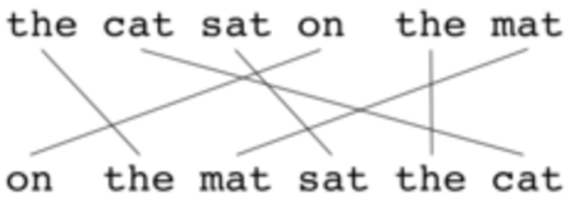
Tad BLEU metriku aprēķina kā modificētu vidējo ģeometrisko no precizitātēm

$$BLEU = BP \left(\prod_{n=1}^{4} p_n \right)^{1/4}.$$

BLEU bieži kritizē, ka šī metrika nepilnīgi izmanto informāciju par referenču garumiem.

5.5.2. METEOR metrika

Tiek apskatīts viens teikums $c \in C$ no kandidāt-kopas un vienu teikumu no referenču kopas $r \in R$. METEOR [17] metrika izmanto piekārtojuma (*alignment*) jēdzienu, kas ir attēlojums, kurš katram vārdam $u \in c$ piekārto to pašu vārdu no references teikuma $u' \in r$. Piekārtojums katram u' var piekārtot tikai vienu vārdu no kandidāt-teikuma c. Piekārtojumu var vizualizēt kā "līniju" definēšanu starp teikumiem 5.2.



5.2. att. Piekārtojuma vizualizācija

Šādi piekārtojumi var būt vairāki. No visiem piekārtojumiem tiek izvēlēts tieši tāds piekārtojums, kurš visvairāk vārdiem u atrod u' un kura definējošās līnijas krustojas vismazāk. Tad ar l tiek apzīmēts tikko minēto līniju skaits izvēlētajam piekārtojumam. Tad tiek definēti lielumi $P = l/|c|$, $R = l/|r|$ un $F = \frac{10PR}{R+9P}$.

Piekārtojums nepieciešams arī lai sadalītu kandidāt-teikumu *fragmentos*, kas nepieciešams lai aprēķinātu "soda" konstanti.

Tiek pieņemts, ka divi vārdi v un u pieder piekārtojuma domēnam, tad šie vārdi atrodas vienā fragmentā, ja:

- u un v atrodas blakus kandidāt-teikumā;
- u un v atrodas blakus references teikumā (nav svarīgi vai secība atšķiras no secības kandidāt-teikumā).

Ar d tiek apzīmēts ar šiem likumiem iegūtais fragmentu skaits. "Soda" konstante tiek aprēķināta kā $e = 0.5\left(\frac{d}{l}\right)^3$. Tad METEOR metriku aprēķina pēc formulas

$$METEOR = (1 - e)F.$$

5.5.3. ROGUE metrika

Lai novērtētu attēlu aprakstu kvalitāti, tika, izmantota arī $ROGUE-L$ metrika, kura balstās uz garākās kopīgās virknes garuma jēdzienu.

Lai $(c_1,...,c_n)$ sauc par virknes $(b_1,...,b_m)$ apakšvirkni, ja eksistē strikti pieaugošu naturālu skaitļu virkne $I = (i_1,....,i_n)$, ka $\forall k \in \{1,...,n\} : c_k = b_{i_k}$. Tad par divu virkņu $a = (a_1,...,a_d)$ un $b = (b_1,...,b_m)$ kopīgo garāko apakšvirkni sauca garāko virkni, $(c_1,...,c_n)$, kas ir gan a, gan b apakšvirkne.

Attiecinot garāko kopīgo apakšvirkni uz teikumiem kā virkņu elementus pieņemot vārdus, ar $LCS(r,c)$ tiek definēts garākās apakšvirknes garums starp teikumiem r un c.

Tad ROGUE-L metriku definē kā

$$\text{ROGUE-L} = \frac{(1 + \left(\frac{|c|}{|r|}\right)^2)RP}{R + \left(\frac{|c|}{|r|}\right)^2 P}.$$

Visas no minētajām mašīntulkošanas metrikām šā darba ietvaros tika implementētas valodā *Python* un pielietotas attēlu aprakstošo modeļu salīdzināšanā [19]. Metrikas tika pielietotas teikumiem, kuru vārdi tika normalizēti ar *stemming* algoritmu implementāciju *Python* bibliotēkā *NLTK*.

6 Mašīnmācīšanās rīks Tensorflow

Nozīmīgu daļu laika, kas tika veltīts šī darba veikšanai, aizņēma rīka *Tensorflow* apguve, bez kura šo darbu veikt nebūtu iespējams. Tāpēc šī nodaļa ir veltīta īsam šī rīka aprakstam un demonstrēšanai. Apraksts balstīts uz materiālu [20].

Tensorflow ir dziļās mašīnmācīšanās rīks, kuru izlaida *Google* 2015. gadā. Šis rīks ir salīdzinoši jauns, taču jau ieguvis plašu popularitāti.

6.1. Modeļa definēšana

Tensorflow programmēšanas interfeisi (API) ir vairākās populārās valodās, kā *Python*, *C++*, *Java* un *Go*. Šī darba ietvaros tika izmantots API valodā *Python*.

No lietotāja perspektīvas centrālais uzdevums ir definēt skaitļošanas grafu, skaitļošanas grafs ietver sevī NN vienādojumus, kļūdu aprēķināšanu un apmācības algoritma pieteikšanu, bet lietotājam nevajag manuāli rakstīt gradienta aprēķināšanas vienādojumus, *Tensorflow* to izdara automātiski.

Kā piemērs tiks parādīts vienkārša NN ar diviem pilnsaistes slāņiem, ar ieejas dimensiju 2 un ar izejas dimensiju 1. Šī īsā pamācība ilustrēs galvenās idejas darbā ar *Tensorflow*.

Vispirms importējam *Tensorflow* bibliotēku *Python* vidē:

```
import tensorflow as tf
```

Skaitļošanas grafs tiek definēts sākot no datu ievades virsotnēm. Tenzorus skaitļošanas grafā, kuri var mainīties skaitļošanas laikā, definē ar *Tensorflow* metodes *placeholder* palīdzību:

```
inputs = tf.placeholder(tf.float32, shape=(None, 2))
outputs = tf.placeholder(tf.float32,shape=(None,1))
```

Tagad tiks definēti tīkla slāņi. Deklarējot slāņus, grupēju tenzorus (skaitļošanas grafa šķautnes) un operācijas (skaitļošanas grafa virsotnes) zem katra slāņa. Modeļa parametrus deklerē ar *tf.Variable*, ir jānorāda komanda, kura ģenerēs sākuma gadījuma vērtības parametriem - šajā gadījumā *tf.randomnormal*.

- Pirmais slānis:

```
with tf.variable_scope("slanis1"):
  svari_1 = tf.Variable(tf.random_normal([2, 15]),name="svari")
  novirzes_1 = tf.Variable(tf.random_normal([1, 15]),name="novirze")
  lin1 = tf.matmul(inputs, svari_1) + novirzes_1
  layer_1_outputs = tf.sigmoid(lin1)
```

- Otrais slānis:

```
with tf.variable_scope("slanis2"):
  svari_2 = tf.Variable(tf.random_normal([15, 3]),name="svari")
  novirzes_2 = tf.Variable(tf.random_normal([1, 3]),name="novirze")
  lin2 = tf.matmul(layer_1_outputs, svari_2) + novirzes_2
  layer_2_outputs = tf.sigmoid(lin2)
```

- Trešais slānis:

```
with tf.variable_scope("slanis3"):
  svari_3 = tf.Variable(tf.random_normal([3, 1]),name="svari")
  novirzes_3 = tf.Variable(tf.random_normal([1, 1]),name="novirze")
  lin3 = tf.matmul(layer_2_outputs, svari_3) + novirzes_3
  outputs_ = tf.sigmoid(lin3)
```

Tiek definēta skaitļošanas grafa virsotne, kura aprēķina kļūdas funkciju (kļūdas funkcija ietver aktivizācijas funkciju, līdz ar to kļūdas funkcijas ieejā norāda summācijas funkciju):

```
error = tf.reduce_mean(
  tf.nn.sigmoid_cross_entropy_with_logits(
  labels = outputs,
```

```
  logits=lin2
))
```

Tiek deklarēta, kura aprēķina vienu gradienta samazināšanas algoritma iterāciju:

```
train = tf.train.GradientDescentOptimizer(0.5).minimize(error)
```

Tagad, kad modelis aprakstīts ar *Tensorflow* skaiļošanas grafu, tas tiks atvērts *Tensorflow* sesijā lai apmācītu modeli.

```
sess = tf.Session()
sess.run(tf.global_variables_initializer())
```

Python ciklā tiek atkārtoti aprēķināta *train* virsotne, kura aprēķina vienu algoritma iterāciju un saglabā parametru jaunās vērtības

```
for i in range(1000):
  sess.run(train)
```

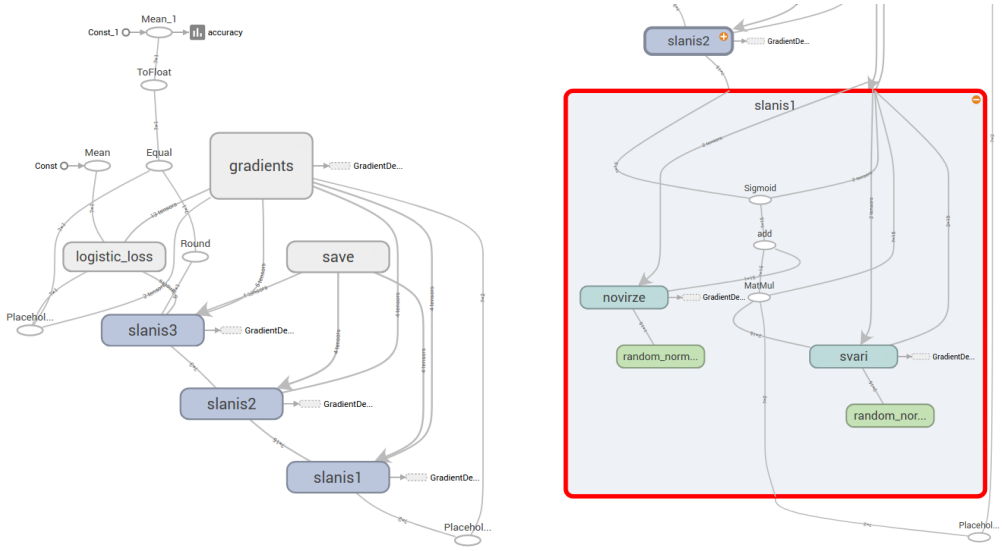
6.2. Gradienta aprēķināšana

Tikko tiek izsaukta skaitļošanas grafa *tf.train...* virsotne skalārs (1-dimensionāls tensors), kurš norādīts funkcijā *minimize(error)*, tiek simboliski atvasināts pēc parametriem. Līdz ar to lietotājam nav jārēķina gradients.

6.3. Vizualizācijas rīks *Tensorboard*

Šajā piemērā apskata salīdzinoši ļoti vienkāršu modeli. Taču praksē neironu tīklu arhitektūras var būt ļoti sarežģītas, tāpēc ir izstrādāts rīks *Tensorflow* skaitļošanas grafu vizualizācijai - *Tensorboad* 6.1.

Tā kā grafs var būt milzīgs, tad vizualizācijas mērogu var mainīt, šajā piemērā ir tuvāk apskatīts pirmais tīkla slānis.



(a) Kopumā (b) Pietuvināts

6.1. att. *Tensorflow* skaitļošanas grafu vizualizācija rīkā *Tensorboad*

6.4. Tensorflow skaitļošanas iespējas

Praksē viens no lielākajiem *Tensorflow* plusiem ir iespēja neironu tīklus novērtēt arī uz GPU. *Tensorflow* ir radīts tā, lai lielus skaitļošanas grafus varētu apstrādāt paralēli uz vairākiem datoriem, kā arī uz GPU un CPU. *Tensorflow* arī atļauj norādīt, uz kuras mašīnas katru operāciju izpildīt un kur glabāt tenzorus.

7 Rezultāti

Visi modeļi tika apmācīti uz autora personīgā datora ar *ubuntu 15.10* operētājsistēmu un šādiem svarīgākajiem atribūtiem:

- CPU - *Intel i5-6500*;
- GPU - *Nvidia GeForceX GT 970*;
- RAM - *16 GB*.

Eksperimentu programmas kodu var iegūt autora *GitHub* repozitorijā [19].

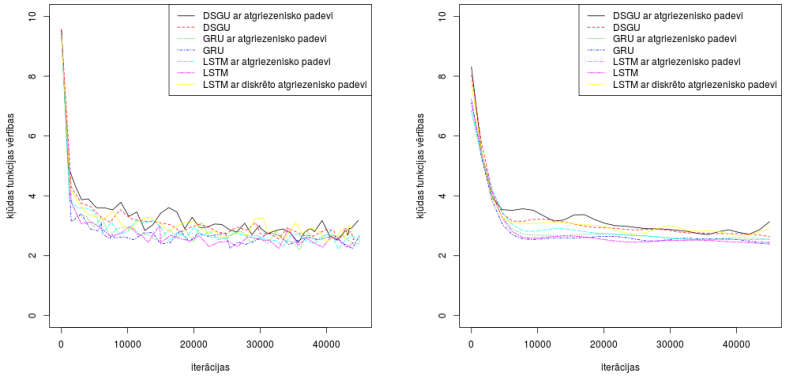
Modeļi tika apmācīti uz MSCOCO [21], kurš sadalīts apmācības/validācijas/testa kopās ar apjomiem 82783/40504/40775, kura katrs elements sastāv no viena attēla un 5 aprakstiem. Līdz ar to apmācībai dati pieauga vēl 5 reizes. Dati aizņēma aptuveni 140 GB atmiņas. Cilvēki, kuri veidoja attēlu aprakstus, centās lai, apraksti būtu pēc iespējas objektīvāki.

7.1. Modeļu apmācības rezultāti

Grafikos 7.1 var apskatīt modeļu kļūdas funkcijas vērtības atkarībā no atpakaļatgriezeniskā algoritma iterāciju skaita.

7.1. tabula. Modeļu kļūdas funkcijas vērtības validācijas datiem

	ar atgriezenisko padevi	parastie	ar diskrēto atgriezenisko padevi
DSGU	13.64533	15.10605	
GRU	10.93070	11.99752	
LSTM	10.85085	11.81248	13.83994



(a) Gludinātas (b) Negludinātas

7.1. att. Kļūdu funkcijas vērtību vērtības atkarībā no iterāciju skaita

Apmācības process tika beigts balstoties uz iterāciju skaitu nevis sasniegto kļūdas funkcijas vērtību. Modeļiem bez atgriezeniskās padeves viena apmācības iterācija aizņēma vidēji 0.4 sekundes, taču modeļiem ar atgriezenisko padevi viena iterācija aizņēma 0.5 sekundes.

7.1.1. Mašīntulkošanas metriku rezultāti

Modeļi tika novērtēti arī ar mašīntulkošanas metrikām BLEU, METEOR un ROGUE-L. Šajā analīzē tika izmantoti 2000 attēli no validācijas datu kopas. Katram attēlam ir attiecīgi 3 kandidāt-tulkojumi un 5 references tulkojumi. Metrikas tika pielietotas katram kandidāta-references pārim (tad kopā sanāk 15), tad kā agregātā metrikas vērtība tika ņemta maksimālā vērtība no iegūtajām 15 metrikas vērtībām.

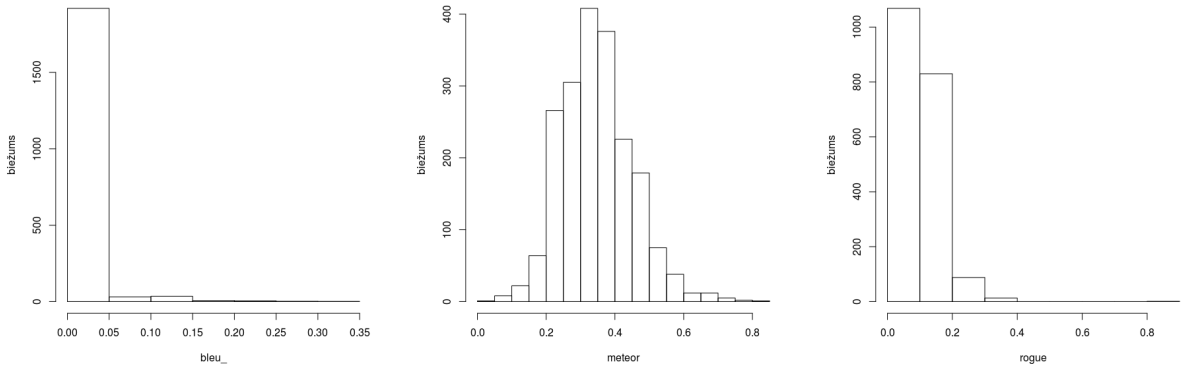
Tā kā katram modelim un metrikai tika iegūtas 2000 metrikas vērtības, salīdzināšanai tika izmantota modelēšana ar GLM modeļu koeficientu nozīmība. Tātad neironu tīklu metriku rezultāti tika novērtēti ar GLM modeli atkarībā no kategoriskā mainīgā "NN arhitektūra" (šajā gadījumā pieņem 7 vērtības, jo tiek salīdzinātas 7 NN arhitektūras) ar bāzes vērtību "LSTM arhitektūra". Šī pēc idejas ir līdzīga ANOVA pieejai, bet tiek pār-

7.2. tabula. BLEU metrikas GLM modeļa koeficienti un p-vērtības

NN	Estimate	Std. Error	t value	Pr(>|t|)
LSTM	0.005	0.0005	9.3926	0
DSGU ar atgriezenisko padevi	0.0019	0.0009	2.1191	0.034
GRU	0.0013	0.0009	1.4627	0.144
GRU ar atgriezenisko padevi	0.0016	0.0009	1.8146	0.07
DSGU	0.0002	0.0008	0.321	0.748
LSTM ar atgriezenisko padevi	0.0028	0.001	2.7892	0.005
LSTM ar diskrēto atgriezenisko padevi	0.0014	0.0009	1.5821	0.114

baudīta hipotēze:"vai modelis A atšķiras no modeļa B", nevis hipotēzi:"metriku vērtības ir atkarīgas no NN arhitektūras". Tika izmantoti GLM modeļi, jo dažu metriku sadalījumi nelīdzinājās normālajam, šādos gadījumos tika lietots gamma sadalījums ar identitātes saites funkciju. Kā mainīgā "NN arhitektūra" bāzes līmenis tika ņemts oriģinālais LSTM modelis, tad jebkura cita modeļa B koeficienta statistiskā nozīmība (pēc t-testa) liecinātu par modeļa B nozīmīgi atšķirīgu rezultātu kā oriģinālajam modelim.

Lai novērtētu šī statistiskā testa pareizību, attēlos 7.2 tiek vizuāli novērtēti metriku vērtību histogrammas.



(a) BLEU vērtību sadalījums (b) METEOR vērtību sadalījums (c) ROGUE-L vērtību sadalījums

7.2. att. Mašīntulkošanas metriku sadalījumi 2000 validācijas datiem

Tabulās 7.2, 7.3 un 7.4 var redzēt metriku GLM modeļu koeficientus. No metriku rezultātiem var novērot, ka LSTM modelim ar atgriezenisko padevi ir vislabākie rezultāti. Turklāt tas nozīmīgi atšķiras no LSTM bez atgriezeniskās padeves.

7.3. tabula. METEOR metrikas GLM modeļa koeficienti un p-vērtības

NN	Estimate	Std. Error	t value	Pr(>|t|)
LSTM	0.2997	0.0024	124.3643	0
DSGU ar atgriezenisko padevi	-0.0001	0.0034	-0.0274	0.978
GRU	-0.0003	0.0034	-0.0771	0.938
GRU ar atgriezenisko padevi	0.0037	0.0034	1.1003	0.271
DSGU	0.0087	0.0034	2.5435	0.011
LSTM ar atgriezenisko padevi	0.0115	0.0034	3.3702	0.001
LSTM ar diskrēto atgriezenisko padevi	0.0032	0.0036	0.901	0.368

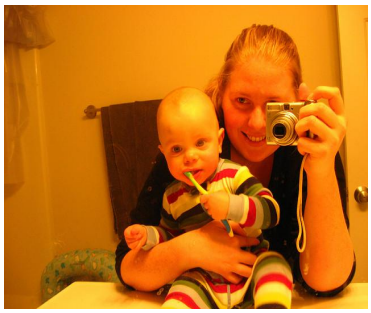
7.4. tabula. ROGUE-L metrikas GLM modeļa koeficienti un p-vērtības

NN	Estimate	Std. Error	t value	Pr(>|t|)
LSTM	0.1084	0.0013	86.563	0
DSGU ar atgriezenisko padevi	-0.0014	0.0018	-0.8229	0.411
GRU	-0.0004	0.0018	-0.2053	0.837
GRU ar atgriezenisko padevi	0.0016	0.0018	0.8771	0.38
DSGU	0.0039	0.0018	2.1471	0.032
LSTM ar atgriezenisko padevi	0.0061	0.0018	3.3667	0.001
LSTM ar diskrēto atgriezenisko padevi	0.0023	0.0018	1.3009	0.193

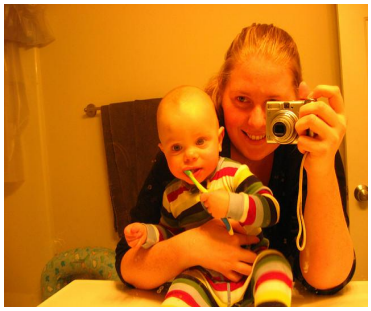
7.1.2. Modeļu attēlu apraksti

Tabulās 7.9, 7.10, 7.7, 7.8, 7.5, 7.6, 7.11 var redzēt modeļu ģenerētos aprakstus izvēlētam bildēm ar sarežģītu saturu. Tabulās ir parādīti modeļu ģenerētie teikumu ar tiem piekārtotajām varbūtībām, kā arī katra modeļa ģenerētie apraksti pēc 15000 apmācības algoritma iterācijām un 45000 iterācijām.

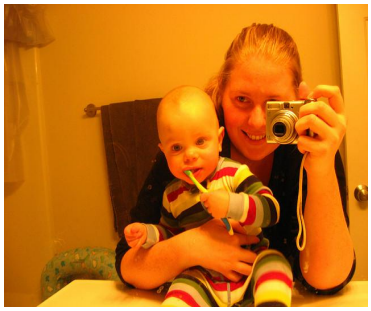
7.5. tabula. Modeļa LSTM attēlu apraksti

attēls	apmācot 15000 iterācijas	apmācot 45000 iterācijas
	• two men are playing a video game together . (p=0.000468) • two men are playing a video game together (p=0.000079) • two men are playing a video game on the wii . (p=0.000037)	• a man and woman playing a video game . (p=0.001003) • a man and a woman playing a video game . (p=0.000910) • a man and a woman are playing a video game . (p=0.000431)
	• a little girl brushing her teeth in a bathroom . (p=0.000372) • a little girl brushing her teeth with a toothbrush . (p=0.000251) • a little girl brushing her teeth in a bathroom (p=0.000071)	• a little girl is brushing her teeth with a toothbrush . (p=0.000357) • a young child is brushing her teeth with a toothbrush . (p=0.000344) • a little girl holding a toothbrush in her mouth . (p=0.000335)
	• a cat sitting on top of a laptop computer . (p=0.006244) • a cat sitting on top of a laptop . (p=0.003385) • a cat sitting on top of a laptop computer (p=0.000414)	• a cat sitting on top of a laptop computer . (p=0.002187) • a black and white cat laying on top of a laptop . (p=0.001800) • a black and white cat sitting on a laptop . (p=0.001304)

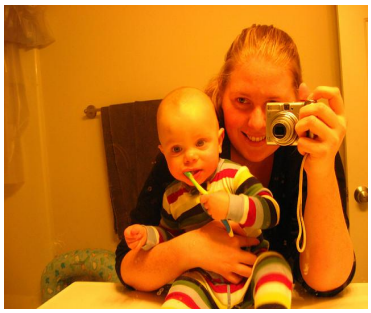
7.6. tabula. Modeļa LSTM ar atgriezenisko padevi attēlu apraksti

attēls	apmācot 15000 iterācijas	apmācot 45000 iterācijas
	• a man and a woman playing a video game . (p=0.000082) • a couple of people playing a video game . (p=0.000081) • a couple of people playing a video game (p=0.000050)	• a man and woman playing a video game . (p=0.000582) • a man and woman playing a video game in a living room . (p=0.000115) • a man and woman playing a video game on a couch . (p=0.000069)
	• a woman holding a toothbrush in his mouth . (p=0.000218) • a woman holding a teddy bear in his mouth . (p=0.000167) • a woman holding a toothbrush in his mouth (p=0.000063)	• a little girl holding a toothbrush in her mouth . (p=0.000156) • a little girl that is holding a toothbrush . (p=0.000146) • a little girl holding a toothbrush in her hand . (p=0.000125)
	• a cat sitting on top of a laptop computer . (p=0.003957) • a black and white cat sitting on top of a laptop computer . (p=0.000869) • a black and white cat sitting on a laptop computer . (p=0.000655)	• a cat sitting on top of a laptop computer . (p=0.010923) • a cat sitting on top of a computer keyboard . (p=0.004311) • a cat sitting on a laptop computer on a desk . (p=0.001967)

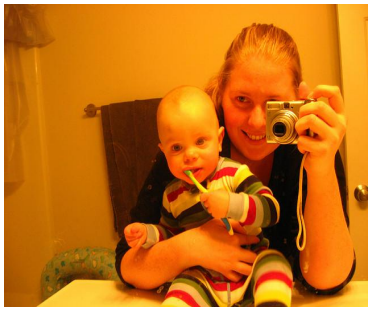
7.7. tabula. Modeļa GRU attēlu apraksti

attēls	apmācot 15000 iterācijas	apmācot 45000 iterācijas
	• a couple of people that are sitting on a couch (p=0.000069) • a couple of people that are sitting on a couch . (p=0.000038) • a couple of people playing a game with nintendo wii (p=0.000012)	• a man and a woman playing a video game (p=0.000229) • a man and a woman playing a video game . (p=0.000209) • a man and a woman are playing a video game (p=0.000084)
	• a young boy brushing his teeth in a bathroom . (p=0.000171) • a young boy brushing his teeth with a toothbrush . (p=0.000094) • a young boy brushing his teeth with a toothbrush (p=0.000083)	• a little boy brushing his teeth with a toothbrush . (p=0.000524) • a baby is holding a toothbrush in her mouth . (p=0.000399) • a baby is holding a toothbrush in its mouth . (p=0.000394)
	• a cat sitting on top of a computer desk . (p=0.003462) • a black cat laying on top of a laptop computer . (p=0.003098) • a black cat laying on top of a computer desk . (p=0.002611)	• a cat sitting on top of a computer keyboard . (p=0.005779) • a cat sitting on top of a laptop computer . (p=0.002840) • a cat sitting on top of a computer monitor . (p=0.001810)

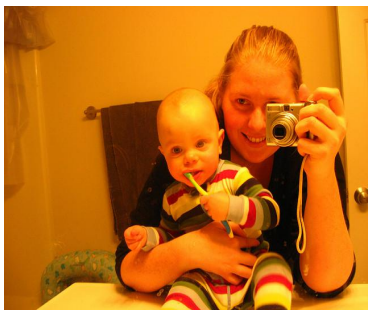
7.8. tabula. Modeļa GRU ar atgriezenisko padevi attēlu apraksti

attēls	apmācot 15000 iterācijas	apmācot 45000 iterācijas
	• a woman is playing video games with her dog . (p=0.000026) • a woman is playing video games with her dog (p=0.000010) • a woman is playing video games in front of her . (p=0.000008)	• a couple of people that are playing a video game (p=0.000524) • a man and woman standing next to each other on a couch . (p=0.000273) • a man and a woman playing a video game (p=0.000148)
	• a baby is holding a toothbrush in her mouth . (p=0.000251) • a baby is holding a toothbrush in his mouth . (p=0.000135) • a baby is wearing a hat is holding a toothbrush . (p=0.000041)	• a little girl sitting in front of a mirror . (p=0.000076) • a little girl sitting in front of a window . (p=0.000073) • a little girl sitting in front of a mirror (p=0.000058)
	• a cat is sitting on a laptop computer . (p=0.002995) • a cat is sitting on the floor next to a laptop computer . (p=0.000333) • a cat is sitting on a laptop computer (p=0.000298)	• a cat sitting on top of a laptop computer . (p=0.015102) • a cat laying on top of a laptop computer . (p=0.012247) • a cat sitting on top of a computer keyboard . (p=0.003395)

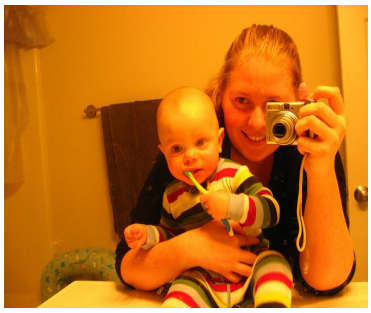
7.9. tabula. Modeļa DSGU attēlu apraksti

attēls	apmācot 15000 iterācijas	apmācot 45000 iterācijas
	• a young girl sitting in a living room in a living room . (p=0.000016) • a young girl sitting in a living room in a room . (p=0.000012) • a young girl sitting in a living room in a room (p=0.000007)	• a little girl sitting in a living room holding a wii controller . (p=0.000006) • a little girl sitting in front of a tv in a living room . (p=0.000003) • a little girl sitting in a living room holding a wii controller (p=0.000003)
	• a young girl sitting in a chair in a room . (p=0.000004) • a young girl sitting in a chair in a bed . (p=0.000003) • a young girl sitting in a chair in a room (p=0.000003)	• a little girl sitting in a chair holding a teddy bear . (p=0.000010) • a little girl sitting in a chair holding a teddy bear (p=0.000007) • a little girl sitting in a chair in front of a mirror . (p=0.000003)
	• a cat sitting on a desk next to a laptop . (p=0.000206) • a cat sitting on a desk next to a laptop (p=0.000068) • a cat sitting on top of a desk next to a laptop . (p=0.000064)	• a cat sitting on top of a laptop computer . (p=0.002449) • a cat laying on top of a laptop computer . (p=0.002424) • a cat laying on top of a laptop computer (p=0.000689)

7.10. tabula. Modeļa DSGU ar atgriezenisko padevi attēlu apraksti

attēls	apmācot 15000 iterācijas	apmācot 45000 iterācijas
	• a man sitting on a couch in a living room . (p=0.000076) • a man sitting on a couch in a living room (p=0.000061) • a man sitting on a couch with a remote (p=0.000051)	• two children playing a game of video game . (p=0.000068) • two children are playing a game of wii . (p=0.000058) • two children are playing a game of video game . (p=0.000036)
	• a man in a red shirt holding a red frisbee (p=0.000000) • a man in a red shirt standing next to a baby (p=0.000000) • a man in a red shirt standing next to a woman holding a baby . (p=0.000000)	• a little girl brushing her teeth with a toothbrush . (p=0.000011) • a little girl holding a teddy bear in her mouth . (p=0.000009) • a little girl holding a teddy bear in a room . (p=0.000006)
	• a cat sitting on top of a laptop computer . (p=0.000411) • a cat sitting on a desk with a laptop . (p=0.000236) • a cat sitting on top of a laptop computer (p=0.000087)	• a cat laying on top of a laptop computer . (p=0.001800) • a cat laying on top of a computer keyboard . (p=0.001341) • a cat laying on top of a laptop computer (p=0.000313)

7.11. tabula. Modeļa LSTM ar diskrēto atgriezenisko padevi attēlu apraksti

attēls	apmācot 15000 iterācijas	apmācot 45000 iterācijas
	• a man is holding a video game controller . (p=0.000132) • a man is holding a video game controller (p=0.000098) • a man and a woman playing a video game (p=0.000071)	• a group of people playing a video game . (p=0.001925) • a group of people playing a video game (p=0.000742) • a group of people playing video games together . (p=0.000486)
	• a little girl brushing her teeth in a bathroom . (p=0.000071) • a little girl brushing her teeth in a bathroom (p=0.000050) • a little girl brushing her teeth with a toothbrush . (p=0.000021)	• a little girl with a toothbrush in her mouth . (p=0.000051) • a little girl with a toothbrush in her mouth (p=0.000045) • a little girl with a toothbrush in her hand . (p=0.000033)
	• a cat sitting on top of a laptop computer . (p=0.003554) • a cat sitting on top of a laptop computer (p=0.002109) • a cat sitting on top of a computer desk . (p=0.000558)	• a cat sitting on top of a laptop computer . (p=0.006802) • a black cat sitting on top of a laptop computer . (p=0.004017) • a black cat sitting on top of a desk . (p=0.001068)

7.2. Apmācības rezultātu iztirzāšana

Grafikos 7.1 var redzēt, ka sākotnējās iterācijās kļūdas funkcija dilst ļoti ātri visiem modeļiem, pēc 10000-ās iterācijas kļūdas funkcija vairs gandrīz nedilst, taču svārstās. Tas varētu liecināt par pārāk liela λ izvēli, bet tā kā λ tiek samazināts atkarībā no iterāciju (epohu) skaita eksponenciāli, tad tam nevajadzētu būt iemeslam.

Visu modeļu kļūdas izturas ļoti līdzīgi, līdz ar to, modeļus atkārtoti apmācot ar jauniem inicializētajiem gadījuma parametriem nozīmīgi zemāki kļūdu līmeņi visdrīzāk netiks iegūti.

Iespējams, ka visu modeļu kļūdu funkcijām globālo minimumu vērtības ir līdzīgas un visi modeļi ir aptuveni sasnieguši savu kļūdu funkciju lokālos minimumus, kuri ir ļoti tuvu (pēc vērtības) globālajam minimumam. Ja šis arguments ir patiess, tad modeļu parametri ir konverģējuši uz ļoti optimālām parametru kombinācijām.

Kļūdas funkcija validācijas datiem 7.1 ir, salīdzinoši ar tās vērtībām uz apmācības datiem, ļoti liela. Līdz ar to varētu tikt secināts, ka visi modeļi cieš no *overfitting* problēmas. Bet, tā kā izmantotie vārdu sinonīmi un teikumu struktūras var būt ļoti atšķirīgas vienam attēlam, tad šo fenomenu nevar tīri norakstīt uz *overfitting* problēmas.

Starp modeļiem ar atgriezenisko padevi un bez ir maza kļūdu atšķirība, kas arī bija gaidīts, jo modeļu ieejas ir neskaidrākas atgriezeniskās padeves dēļ. Ļoti iespējams, ka palielinot apmācības iterāciju skaitu, modeļiem ar atgriezenisko padevi kļūdu funkcijas vērtības būtu ļoti līdzīgas kā bez atgriezeniskās padeves.

7.3. Attēlu aprakstu iztirzāšana

Aplūkotie attēli tika izvēlēti nejauši, kā arī šie attēli nav "vienkārši", jo visi satur objektu un relāciju kombinācijas, kuras nav izteikti tipiskas.

Attēlā, kur draugu grupa spēlē videospēli, GRU ar atgriezenisko padevi aprakstīja pareizi, kā arī LSTM ar atgriezenisko padevi aprakstīja pareizi pie 15000 iterācijām, ar 45000 atkal neprecīzi. Interesanti, ka daži modeļi ģenerētajos teikumos ir pieminējuši arī *Nintendo Wii* video spēli. DSGU ar atgriezenisko padevi personas novērtēja kā bērnus.

Bildē, kur māte savam mazulim mazgā zobus un to ir uzņēmusi ar fotokameru spogulī, neviens modelis neaprakstīja pareizi. Gandrīz visi attēla objekti un relācijas ir pieminētas atsevišķi - mazulis/jauns bērns, zobu mazgāšana, turēšana, spogulis. Taču neviens modelis

nespēja apvienot visu vienotā aprakstā.

Attēlu, kur melns kaķis ir apgūlies blakus klēpjdatoram, visi modeļi aprakstīja ļoti līdzīgi - ka kaķis atrodas uz datora, kas nav pilnīgi pareizi. Daudzi modeļi piemin arī kaķa krāsu.

Autors apskatīja arī daudzu citu attēlu aprakstus. Vienkāršākus attēlus, kur, piemēram, viena persona nodarbojas ar kādu sporta veidu, gandrīz visi modeļi aprakstīja pareizi. Attēli, kuros attēlots viens objekts, kurš dara sev tipisku darbību, gandrīz vienmēr tiek pareizi aprakstīti.

7.4. Mašīntulkošanas metriku rezultātu iztirzāšana

Atgriezeniskās padeves pielietošanu var attaisnot nelielu, taču nozīmīgu uzlabojumu mašīntulkošanas metriku rezultātos. Tā kā, pielietojot šo tehniku apmācības laiks pieaug par $\sim 25\%$, tad iespējams atgriezeniskās padeves rezultāti tiktu apsteigti vienkārši neironu tīklus apmācot ilgāk.

Interesanti, ka neironu tīklu arhitektūru sadalījums pēc kļūdu funkciju vērtībām validācijas datiem, neatbilst sadalījumam pēc mašīntulkošanas metrikām. Jo kļūdas funkcijas liek uzsvaru uz tieši tādu pašu teikumu ģenerēšanu, bet mašīntulkošanas metrikas liek uzsvaru uz dažādu n-grammu atkārtošanos starp kandidāt-teikumiem un referencēm.

Vēl svarīgi piebilst, ka šajā darbā nav pētīts kā minētie rādītāji un metrikas izturas atšķirīgiem sākuma parametriem, kuras tiek ģenerētas pēc gadījuma principa atpakaļatgriezeniskā algoritma inicializēšanai. Modelis netika apmācīts tika daudz iterāciju kā tas aprakstīts darbā [14] autora ierobežoto skaitļošanas resursu dēļ.

Secinājumi

Darbā tika apskatītas dziļo neironu tīklu arhitektūras attēlu aprakstīšanai. Vispirms tika apskatīta izmantoto modeļu teorija. Teorijas apskats sakās ar vispārīgu neironu tīklu iepazīstināšanu un to apmācību, tālāk tika tuvāk apskatītas neironu tīklu arhitektūras, kuras izmantotas attēlu aprakstīšanas modelī.

Tika apmācīts 2015. gada MSCOCO attēlu aprakstīšanas sacensības uzvarējošais modelis, izmantojot jau gatavu implementāciju, tad tika veiktas gan tīkla arhitektūras izmaiņas, gan izmaiņas apmācības algoritmā. Tika eksperimentēts ar attēlu aprakstu modeļa teikumu ģenerējošo daļu, pielietotas LSTM, GRU un DSGU rekursīvo tīklu arhitektūras. Apmācības algoritms tika modificēts ar atgriezenisko padevi un diskrēto atgriezenisko padevi.

Salīdzinot attēlu aprakstošos modeļus, oriģinālais modelis uzrādīja viszemāko apmācības datu un testa datu kļūdu. Taču modeļus salīdzinot ar mašīntulkošanas metrikām labākais izrādījās modelis ar LSTM un atgriezenisko padevi. Līdz ar to sākotnējais mērķis - uzlabot attēlu aprakstošo modeli - izdevās.

Literatūra

[1] Jānis Zuters. *Neironu Tīkli*. Latvijas Universitāte.

[2] Eva Zerz, Uwe Helmke, Dieter Pratzel-Wolters. *Mathematial Theory of Neural Networks.* Techincal University of Kaiserslautern, University of Wurzburg, 2001.

[3] R. Rojas. *Neural Networks*, Springer-Verlag, Berlin, 1996.

[4] Geoff Gordon, Ryan Tibshirani. *10-725 Optimization Fall 2012, Lecture 5: Gradient Desent Revisited*. School of Computer Science, Carnegie Mellon University.

[5] Herbert Jaeger. *A tutorial on training recurrent neural networks*. Fraunhofer Institute for Autonomous Intelligent Systems (AIS), International University Bremen, 2013.

[6] Hava T. Siegelmann, Eduardo D. Sontag. *On Computational Power Of Neural Nets*.

[7] Razvan Pascanu, Tomas Mikolov, Yoshua Bengio. *On the difficulty of training recurrent neural networks*, 2013.

[8] Sepp Hochreiter, Jurgen Schmidhuber. *LONG SHORT-TERM MEMORY*. Neural Computation 9(8):1735-1780, 1997.

[9] Felix Gers. *Long Short-Term Memory in Recurrent Neural Networks*. Swiss Federal Institute of Technology in Lausanne, 2001.

[10] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio. *Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation*.

[11] Yuan Gao, Dorota Glowacka. *Deep Gate Recurrent Neural Network*. University of Helsinki, 2016.

[12] David Stutz. *Seminar Report: Understanding Convolutional Neural Networks*, 2014.

[13] MD. Rayed Bin Wahed, Akm Nivrito. *Comparative Analysis between Inception-v3 and Other Learning Systems using Facial Expressions Detection*. Brac University, 2016.

[14] Oriol Vinyals, Alexander Toshev, Samy Bengio, Dumitru Erhan. *Show and Tell: Lessons learned from the 2015 MSCOCO Image Captioning Challenge*.

[15] "Show and Tell: A Neural Image Caption Generator" Pieejams: https://github.com/tensorflow/models/tree/master/im2txt

[16] Xingyi Song, Trevor Cohn, Lucia Specia. *BLEU deconstructed: Designing a Better MT Evaluation Metric*. University of Sheffield.

[17] Satanjeev Banerjee, Alon Lavie. *METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments*. Language Technologies Institute Carnegie Mellon University.

[18] Chin-Yew Lin. *ROUGE : A Package for Automatic Evaluation of Summaries*. Information Sciences Institute University of Southern California.

[19] "Maģistra darba programmas" Pieejams: https://github.com/BOpermanis/img2txt_project

[20] Peter Goldsborough. *A Tour of TensorFlow*. Technische Universität München, 2016.

[21] Tsung-Yi Lin, James Hays, Michael Maire, Serge Belongie, Pietro Perona , Deva Ramanan, Lubomir Bourdev, Ross Girshick, C. Lawrence Zitnick, Piotr Dollár. *Microsoft COCO: Common Objects in Context*, 2015.

[22] Yoshua Bengio, Aaron Courville, Pascal Vincent. *Representation Learning: A Review and New Perspectives*, 2014.

[23] "Vector Representations of Words" Pieejams: https://www.tensorflow.org/tutorials/word2vec

[24] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán. *Translating Embeddings for Modeling Multi-relational Data*.

[25] Guntis Bārzdiņš, Kārlis Čerāns, Renārs Liepiņš. *Dziļā mašīnmācīšanās*. Latvijas Universitāte.

[26] Omer Levy, Yoav Goldberg. *Linguistic Regularities in Sparse and Explicit Word Representations*. Bar-Ilan University.

[27] Laurens van der Maaten, Geoffrey Hinton. *Visualizing Data using t-SNE*, 2008.

Maģistra darbs “Attēlu aprakstīšana ar konvolūciju un rekurentajiem neironu tīkliem” izstrādāts LU Fizikas un Matemātikas fakultātē.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: Bruno Opermanis

_____ _____

(paraksts) (datums)

Rekomendēju darbu aizstāvēšanai.
Vadītājs: prof. Dr. dat. Guntis Bārzdiņš

_____ _____

(paraksts) (datums)

Recenzents: prof. Dr. math. Jānis Buls

_____ _____

(paraksts) (datums)

Darbs iesniegts Matemātikas nodaļā _____

(datums)

_____

(darbu pieņēma)

Darbs aizstāvēts maģistra gala pārbaudījuma komisijas sēdē

_____ prot. Nr. _____, vērtējums_____

(datums)

Komisijas sekretārs/-e: _____ _____

(Vārds, Uzvārds) (paraksts)