VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

POPIS FOTOGRAFIÍ POMOCÍ REKURENTNÍCH NEURONOVÝCH SÍTÍ

IMAGE CAPTIONING WITH RECURRENT NEURAL NETWORKS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. JAKUB KVITA
AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAL HRADIŠ, Ph.D.
SUPERVISOR

BRNO 2016

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2015/2016

Zadání diplomové práce

Řešitel: **Kvita Jakub, Bc.**

Obor: Počítačová grafika a multimédia

Téma: **Popis fotografií pomocí rekurentních neuronových sítí**
Image Captioning with Recurrent Neural Networks

Kategorie: Zpracování obrazu

Pokyny:

1. Prostudujte základy neuronových sítí a back-propagation.
2. Vytvořte si přehled o současných metodách pro generování popisků na základě fotografií pomocí rekurentních sítí.
3. Navrhněte konkrétní metodu pro generování popisů fotografií pomocí rekurentních sítí.
4. Obstarejte si databázi vhodnou pro experimenty.
5. Implementujte navrženou metodu a proveďte experimenty nad datovou sadou.
6. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.
7. Vytvořte stručné video prezentující vaši práci, její cíle a výsledky.

Literatura:

- Vinyals et al.: Show and Tell: A Neural Image Caption Generator. CVPR 2015.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese
http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Hradiš Michal, Ing., Ph.D.**, UPGM FIT VUT

Datum zadání: 1. listopadu 2015

Datum odevzdání: 25. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato práce se zabývá automatickým generovaním popisů obrázků s využitím několika druhů neuronových sítí. Práce je založena na článcích z MS COCO Captioning Challenge 2015 a znakových jazykových modelech, popularizovaných A. Karpathym. Navržený model je kombinací konvoluční a rekurentní neuronové sítě s architekturou kodér–dekodér. Vektor reprezentující zakódovaný obrázek je předáván jazykovému modelu jako hodnoty paměti první LSTM vrstvy v síti. Práce zkoumá, jestli je model s takto jednoduchou architekturou vhodnější pro jiné jazyky než je angličtina, než ostatní současné modely.

Abstract

In this paper we deal with automatic generation of image captions by using multiple types of neural networks. Thesis is based on the papers from MS COCO Captioning Challenge 2015 and character language models, popularized by A. Karpathy. Proposed model is combination of convolutional and recurrent neural network with encoder–decoder architecture. Vector representing encoded image is passed to language model as memory values of first LSTM layer of the network. This work investigate, whether model with such simple architecture is more suitable for languages different than English than other contemporary solutions.

Klíčová slova

rekurentní neuronové sítě, RNN, konvoluční neuronové sítě, CNN, popisování obrázků, LSTM, GRU, MS COCO, Torch, hluboké učení

Keywords

recurrent neural networks, RNN, convolutional neural networks, CNN, image captioning, LSTM, GRU, MS COCO, Torch, deep learning

Citace

KVITA, Jakub. Image Captioning with Recurrent Neural Networks. Brno, 2016. 38 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Hradiš Michal.

Image Captioning with Recurrent Neural Networks

Declaration

I hereby certify that this thesis is a presentation of my original research work and I have excercised reasonable care to ensure it does not to the best of my knowledge breach any law of copyright. Wherever contributions of others are involved, every effort is made to indicate this clearly, with due reference to the literature, and acknowledgement of collaborative research and discussions. The work was done under the guidance of Michal Hradiš at the Brno University of Technology.

....
Jakub Kvita
April 8, 2016

Acknowledgements

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures"(CESNET LM2015042), is greatly appreciated.

Contents

1 Introduction **4**

2 Neural networks **5**
2.1 Feed-forward neural nets 5
2.2 Recurrent neural nets 7
2.2.1 Recurrent architectures 8
2.2.2 Modeling languages 11
2.3 Convolutional neural nets 12

3 Image caption generation **15**
3.1 Related Work 15
3.2 Datasets 20
3.3 Evaluation 21
3.3.1 Automated metrics 22

4 Model design **23**
4.1 General architecture 23
4.1.1 Convolutional net 23
4.1.2 Language model 23
4.2 Learning algorithm and training 23

5 Implementation **25**
5.1 Tools 25
5.1.1 Torch 26
5.2 File structure and workflow 27

6 Experiments and results **28**
6.1 Language model initialization variations 28
6.2 Evaluation 28
6.3 Analysis of results 28

7 Conclusion **29**

Bibliography **30**

List of Abbreviations **35**

Appendices **36**
List of Appendices 37

A CD Content **38**

List of Figures

2.1 Nonlinear functions used in neural nets. 6
2.2 Applying dropout to a neural network. 7
2.3 Unrolling of a recurrent neural net. 8
2.4 Variation of a LSTM. 9
2.5 Variation of a GRU. 10
2.6 Architecture of famous CNN *LeNet-5*. [36] 13

3.1 *Show and Tell* image captioning model. [54] 17
3.2 *From Captions to Visual Concepts* caption generation pipeline. [14] 18
3.3 Examples of attending the correct object while generating the word. [57] . 19
3.4 We can use attention to understand, what was the model looking at, while generating wrong caption. [57] 19
3.5 Task-specific instantiations of the LRCN model. [11] 20

4.1 Architecture of the proposed model. 24

Chapter 1

Introduction

Prepsat.

When does a machine understand an image? One definition could be the following sentence: *A machine understand an image, when in can describe important content of the image.* This description should include present objects, their attributes and relation to each other. Determining the important content of the image can be quite difficult, even for humans, which have been trained for this task since they were born. However, deep learning techniques are proving to be quite successful in this kind of tasks. Similarly to people, these models require large amounts of training data, but later they can evaluate correctly even yet unseen situations.

Deep machine learning, sometimes under the name of neural networks[1], is branch of machine learning based on composing multiple non-linear functions to solve the task. As it is fundamentally different from the standard computer algorithms, it perform well on problems, which are unsuitable these algorithms. For example, neural networks have excellent performance in recognizing speech and images, writing stories and composing music. This work focuses on generating image descriptions in regular English sentences, which is also a task suitable for deep learning.

First, in chapter 2, I will first introduce neural networks and several key concepts which are used later in this work. In chapter 3 current state-of-the-art in the field of image captioning and summary of the key papers will be presented. I will list most popular frameworks and libraries for implementation of deep learning models and mention my experiments with them in the chapter 5.1. Using knowledge of previous chapters I will propose an image captioning model in chapter 4. Expectations of this proposal will be discussed in the concluding chapter 7.

V opravdu dobrém textu mezi nadpisy bude vždy nějaký text a to ne jen jeden řádek. Před každým informačním blokem (kapitolou, podkapitolou) může být shrnutí co se čtenář dozvěděl v minulém textu a uvedení, proč právě teď je ten správný okamžik se pustit do následujícího tématu, jak navazuje na předchozí, jak čtenáře posune v pochopení celého řešení a proč se na následující informace má těšit.

[1] Terms *deep learning* and *neural networks* will be used in this work interchangeably.

Chapter 2

Neural networks

General idea of artificial neural networks emerged after World War II. Perceptron, a single artificial neuron, was created in 1958 by Frank Rosenblatt [47], but it became popular only after combination with the backpropagation algorithm [5, 55]. At that time neural nets have not reached massive popularity, not because they do not work, but because small computing power of machines back then, and also the lack of datasets. Recently (after 2000), neural nets became popular again under the name of "deep learning"to emphasize the use of several layers stacked on top of each other to create deep architectures, which are far more practical than shallow ones. During this reinvention, neural nets have been successfully applied in multiple fields like computer vision [21], speech recognition [18], and natural language modeling [40].

Nowadays, various useful architectures, techniques and applications of neural nets are introduced almost every day. As it is not possible to through all of them, in this chapter I will describe only a handful, which are most significant and will be later used in the research. This chapter is divided into three sections, each focusing on different type of neural nets – feed-forward, recurrent, and convolutional. However, do not see this division as strict and separating, tools introduced in one part can and will be used in different types of networks.

2.1 Feed-forward neural nets

Feed-forward networks are simplest architecture of neural nets, yet they can solve many real world tasks. Most commonly used in classification problems, feed-forward nets showed very promising results, which later proved to be true. Later, they have been replaced by convolutional nets, which are specific type of complex feed-forward neural net. However, simple architectures still have place for utilization.

In this part I will cover linear neuron, rectifiers and other nonlinear functions used, and dropout, as they are most important to the following work. Other tools like softmax layer, loss functions, and training algorithms will be skipped.

Linear unit

In this type of neuron, the output of the unit is simply the weighted sum of its inputs added to a bias term, described by equation

$$y = Wx + b. \tag{2.1}$$

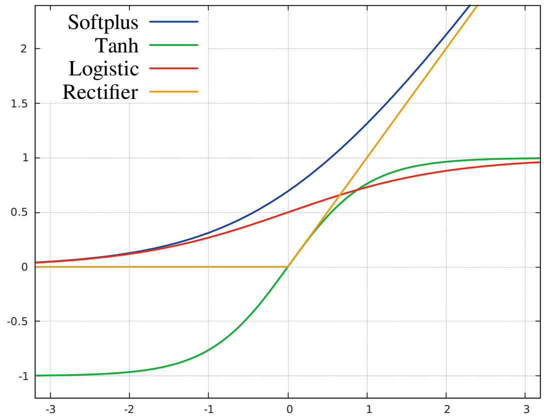

Figure 2.1: Nonlinear functions used in neural nets.

A combination of these neurons performs a linear transformation of the input vector. Ability to perform only linear and affine transformations is also its weakness, as some kind of nonlinear function needs to be added to produce more complicated functions. However it is useful at the beginning and end of the network, to emphasize important features of the input or output and change its dimensionality.

This type of unit is the most basic one. It was part of the Rosenblatt's perceptron [47] as well as the boolean function, which later evolved into nonlinear functions, like Rectifier described further.

Rectifier and ReLU

Combination of linear layers in neural network can result only in another linear layer, which is useless for example on problems of nonlinear separation. To break free from limitations induced, we need to introduce some kind of nonlinearity directly into the network. Most commonly used method is to apply a nonlinear activation function to the output of a linear neuron. As to which function, there are many suitable options, rectifier nowadays being the most popular one.

In the context of neural networks, the rectifier is an activation function defined as

$$f(x) = max(0, x). \tag{2.2}$$

Rectifier is usually used after a linear unit creating together Rectified Linear Unit (ReLU), which showed improvements in restricted Boltzmann machines [43], speech processing [59], and it is also default option in convolutional networks. This unit has several advantages against other functions – in randomly initialized networks, only about 50% of units are activated. There are no problems with vanishing gradient in large inputs. Computation of the function is also more efficient than other functions. Issue with this function is non-differentiability at zero, however it is differentiable at any point arbitrarily close to 0 and can be replaced with softplus [12], which is analytic function smoothly approximating rectifier. Currently, more variations of ReLU were introduced – Leaky ReLU, parametric ReLU, etc. and their performance can be even better [56] than vanilla ReLUs.

Before ReLU, popular functions were hyperbolic tangens and standard logistic function. However, these functions are costly to compute, even though they can be replaced with

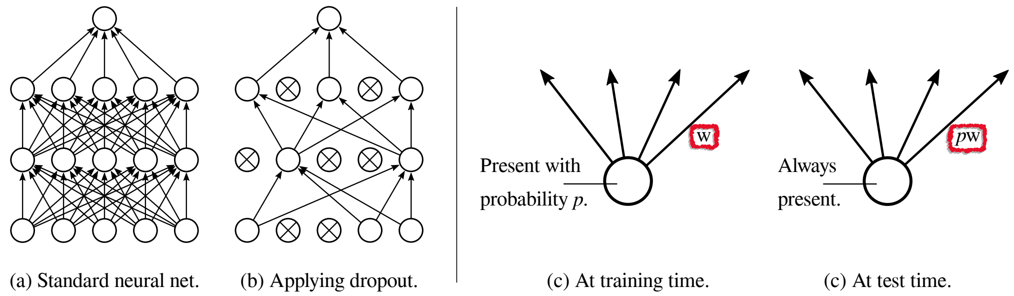

Figure 2.2: Applying dropout to a neural network.

polynomials. Hyperbolic tangens was preferred as better version of logistic function [37]. See how the discussed functions look at Figure 2.1.

Dropout

Dropout [22, 49] can be considered as one of the biggest recent inventions in the field of neural networks. It is extremely simple and effective technique addressing the problem of overfitting. It can be seen as type of regularization, together with techniques like L1 and L2 regularization, and constraining maximum value of weights.

Dropout works with the idea of "dropping out" some of the unit activations in a layer, that is setting them to zero, during training. This can be interpreted as sampling a neural network from the full neural network, and only updating the parameters of the sampled network for the given data. Visualisation is on Figure 2.2, parts *a* and *b*. Dropout behaves differently during sampling phase – all the units are present, but their outputs are multiplied by the same probability used before for dropping them out. See the part *c* and *d* of Figure 2.2.

This technique should prevent complex co-adaptations, in which unit is only helpful in the context of several other specific units. Each neuron instead learns to detect a feature generally useful for computing the answer.

2.2 Recurrent neural nets

Feedforward neural nets are extremely powerful models, but they can be only applied to problems with inputs and outputs of fixed dimensionality. This is a serious drawback, as many of the real-world problems are defined as sequences with lengths that are unknown to us beforehand. Recurrent neural networks were introduced soon after feed-forward nets and they proved to be very useful in this kind of a task. There is vast amount of recurrent neural network types, many not suitable for sequential tasks, like Hopfield networks [26], which are very successful in what they do, but nevertheless not useful for us now.

Apart from classification, which can be more precise when using sequences, one of the most important tasks is next value prediction. This core task can be then extended very simply to predict arbitrary number of future values. Prediction problems are all around us, from the weather forecast and stock market prediction to the autocomplete in smartphones or web browsers. The image captioning problem, which is the main topic in this work, includes the prediction (or generation) task too, as the caption is generated one character at a time starting from the "base caption". More detailed description is further in the

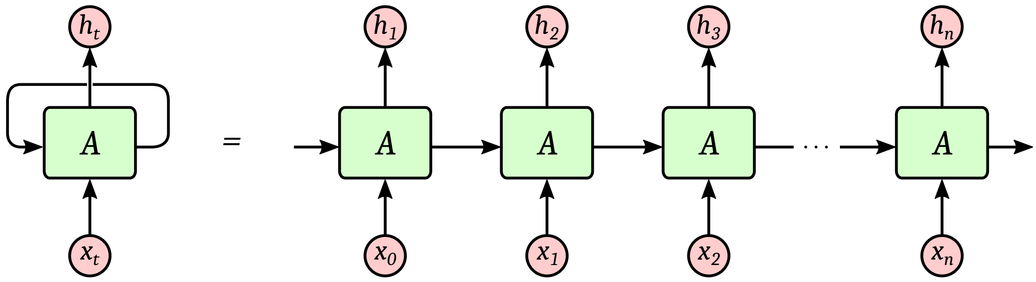

Figure 2.3: Unrolling of a recurrent neural net.

following chapters.

We can interpret recurrent neural network as very deep forward net with shared weights and same inputs and outputs as before. This process of reinterpreting the network is called RNN unrolling and is visualised in Figure 2.3. Layers of this very deep net spread forward in time, together with the input sequence. This is very innovative idea, which enabled training RNN with backpropagation through time, as we are not bound by time during training.

Unrolling the networks on long sequences and creation of very deep neural nets caused manifestation of the vanishing gradient problem, which is one of the most important issues in RNNs. The problem occurs during training with gradient-based learning methods like backpropagation. The chain rule in backpropagation multiplies gradients to compute updates of weights and the traditional activation functions like hyperbolic tangent have gradient in the range of $(-1, 1)$, are the reasons why, the weights updates decrease exponentially while approaching front layers of the network. The vanishing gradient problem was formally identified by Hochreiter in 1991 [23], who, after further research, also proposed one of the solutions to this problem in the form of Long Short-Term Memory.

Long Short-Term Memory unit and other architectures commonly used in RNNs are discussed in following section 2.2.1. Second half (**??**) of this part explains how to process text and model languages for applications with RNNs.

2.2.1 Recurrent architectures

RNNs have many different architectures, however, most of them are derived from the basic fully recurrent network. This network do not have units separated into layers, as each of them has a directed connection to every other unit. Rest of the architectures are special cases of this one, as they group neurons into layers and implement only a subset of the connections. Examples of these architectures can be Hopfield [26] and Elman [13] networks, and Restricted Boltzmann Machines [48]. Different architectures are trying to connect RNN with an external memory resource, which can be a tape in case of Neural Turing Machines [19], a stack in Neural network Pushdown Automata [50], etc. During training RNN unrolling can be applied to these architectures, although training can be quite difficult, as explained earlier.

From here on in I will focus on an architecture called Long Short-Term Memory and architectures derived from it, as they are very powerful and dominating the current field.These units are carefully designed with the vanishing gradient problem in mind and perform better than most of the other architectures.

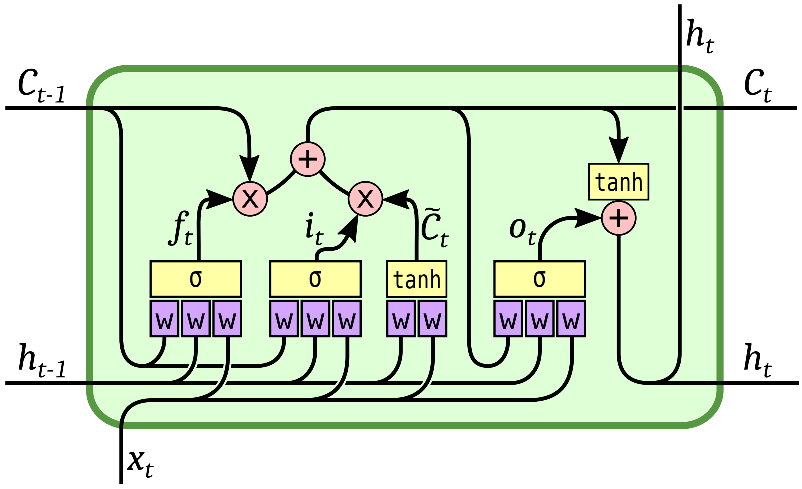

Figure 2.4: Variation of a LSTM.

Long Short-Term Memory

Long Short-Term Memory (LSTM) is a special type of recurrent network, able to learn long-term dependencies. This architecture was introduced by Hochreiter and Schmidhuber [24] after prior research of vanishing gradient problem, and later refined and popularized by other researchers [15, 16].

LSTM was designed to remember a value for an arbitrary length of time. It contains gates that determine, when the input is significant enough to remember, when it should keep or forget the value, and when it should copy the value to the output. To understand the flow of data, see the diagram of LSTM on Figure 2.4. All the gates can be described by the following equations:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}C_{t-1} + b_i) \tag{2.3}$$
$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}C_{t-1} + b_f) \tag{2.4}$$
$$\widetilde{C}_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{2.5}$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \widetilde{C}_t \tag{2.6}$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}C_t + b_o) \tag{2.7}$$
$$h_t = o_t \odot \tanh(C_t) \tag{2.8}$$

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.9}$$

In each time slice LSTM is using current input x_t, last cell state C_{t-1} and unit output h_{t-1} to compute next cell state C_t and output h_t. Variables i_t, f_t, o_t denote values of in following order input, forget and output gates, which are used to control the information flow. LSTM based on these equations is using total of 11 weight matrices, 4 bias vectors, and a standard logistic function σ defined in Equation (2.9). The operation $\odot$ denotes element-wise vector product.

Equations (2.3) to (2.9) are not the only way to create an LSTM unit, they are a variation, which was used for implementing the proposed model. As LSTM is very popular,

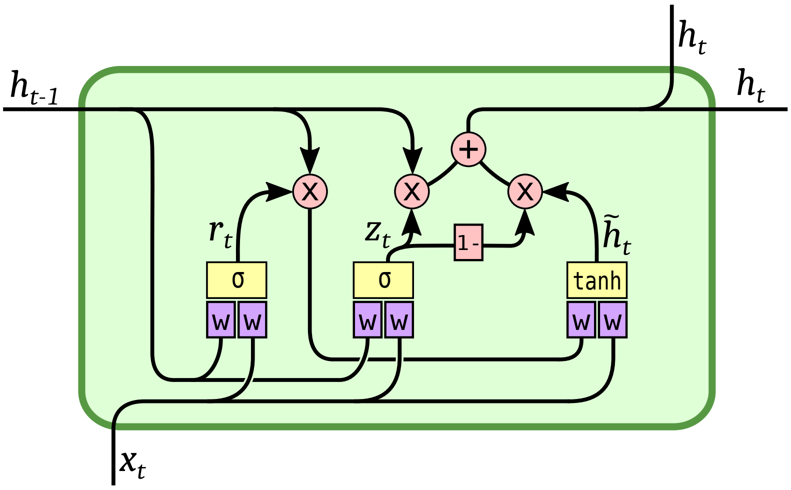

Figure 2.5: Variation of a GRU.

many different forms were created. For example, original LSTM from 1997 [24] did not included the "peephole connections" $W_{ci}C_{t-1}$, $W_{cf}C_{t-1}$, and $W_{co}C_t$. These were added later in work of Gers and Schmidhuber [15]. Another change is to couple input gate i_t and forget gate f_t. Instead of separately deciding what to forget and when to input new information, unit only forgets the value when something new is placed in its place. More units based on LSTM and their comparison is in work of Greff, et al. [20].

Training of the LSTM based network can be performed effectively by standard methods like stochastic gradient descend in the form of backpropagation through time. Major problem with vanishing gradients during training described earlier is not an issue as backpropagated error is fed back to each of the gates.

Gated Recurrent Unit

Gated Recurrent Unit (GRU) [7] is slightly more dramatic variation on the LSTM theme. It combines hidden state of the unit h_t with the saved value C_t, merges input and forget gates into one update gate z_t and some smaller changes. Compare GRU diagram on Figure 2.5 with the previous LSTM figure. GRU is based on following set of equations:

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \tag{2.10}$$
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \tag{2.11}$$
$$\widetilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(h_{t-1} \odot r_t) + b_h) \tag{2.12}$$
$$h_t = (1 - z_t) \odot \widetilde{h}_t + z_t \odot h_{t-1} \tag{2.13}$$

On top of the Equations (2.10) to (2.13), GRU is using the standard logistic function σ defined in Equation (2.9). The operation $\odot$ again denotes the element-wise vector product. As the unit is using only 4 weight matrices, 3 biases and 1 state variable, researchers studied whether it can achieve the performance on same level as previous LSTM.

In Chung's study [8], different types of recurrent units were compared on the polyphonic music datasets. LSTM and GRU were performing significantly better than all of the other architectures, with GRU slightly in the lead. According to Greff, et al. [20] on the other hand, GRU is an average variation, slightly better than vanilla LSTM, with much simpler architecture.

Jozefowicz's study [30] tried to determine whether the LSTM architecture is optimal and if such architecture exists. On variety of tasks and the data GRU outperformed LSTM on all tasks with the exception of language modeling. On top of that, they identified architectures that outperforms both LSTM and GRU. These architectures were found by evolutionary algorithm working on candidate architectures represented by the computational graph. In this work I will explore different type of language modeling than the one used in Jozefowicz's paper [30] and I will not focus on these new types of units. Interestingly they also confirmed that LSTM nearly matched the GRU's performance, when its forget gate bias was initialized to a large value such as 1 or 2, and not to naive initialization around 0. This idea was already presented by Gers [16] and can be interpreted in a way that LSTM will not explicitly forget anything until it has learned to forget.

Generally, researchers agree that most of the LSTM variations, including GRU, are roughly on the same performance level. As the changes introduced in GRU are simplifying the standard LSTM model even though keeping the performance level, GRU has been growing increasingly popular.

2.2.2 Modeling languages

With the addition of LSTM, recurrent neural nets quickly showed great potential in many different types of sequence processing like speech recognition, signal prediction and modeling languages. These result were further improved when researchers started stacking LSTMs on top of each other. Language modeling has several ways to process input text and feed it to the network. In this chapter, I will describe word and character level models, which are most commonly used.

Word level representation of the text is used by most of the state-of-the-art models, have been enhanced by many features and proved very effective for English. In this method, each word is encoded to a vector of a constant length. The neural network then works only with these encodings and does not have direct access to the word and its form. The advantage of this approach is no need to teach the model exact spelling of the words, which also means the model is not going to be confused by homographs[1]. The benefit also is that encoded sequences are much shorter than sequences based on dividing text by character. On the other hand, the disadvantage of this approach is that modeling non-word text, like punctuation and long numbers, can be complicated.

All that is left, is to decide specific encoding for model to use. Simple way that comes to mind is one-hot[2] or one-from-k encoding, which has its advantages, however, there are several issues with it in this application. The task vocabulary often exceeds 100 000 records, which means each input vector would be incredibly long. So long, issues with time complexity of computations would appear. Luckily set of techniques called *word embedding* were developed.

Word embedding [3] is a tool for mapping words or phrases from the vocabulary to suitable vectors of real numbers in low dimensional space (around 200 – 500 dimensions)

[1] A *homograph* is a word that shares the same written form as another word but has a different meaning.
[2] One-hot encoded vector has exactly one high ('1') value and all the others low ('0').

relative to the vocabulary. For example,

$$W(horse) = (0.2,\ 0.4,\ 0.7,\ 0.1,\ \ldots), \tag{2.14}$$
$$W(window) = (0.0,\ 0.6,\ 0.1,\ 0.9,\ \ldots). \tag{2.15}$$

Vectors are usually randomly initialized and then trained to capture structure of the input data to perform some task. An example can be skip-gram model [39], which mapped 783 millions words to vectors of 300 real numbers, while creating reasonable relationships between them. Word embeddings show many interesting properties, like encoding analogies between words as differences between their vectors [41], but I am not going to cover them in more detail in this work.

Character level modeling has been considered as an alternative to word-level, but so far had worse performance. Regardless, it is still considered as an option, because of the much simpler representation of the input and output. Consider roughly 45 characters in English text and over 100000 words created from them. Same input can be modeled in character level by simple one hot encoding, instead of creating whole field of the word embeddings. These models are also more suited for Czech, Russian, and other fusional[3] languages, which heavily use prefixes and suffixes to create new words.

Character level models usually have smaller vocabulary size and tend to take more time for training, as they need to learn spelling of the words and structure of a sentence, on top of the same features of word level. However, with the properly trained character level model, we can benefit from its greater generative abilities, on top of the very simple input and output of the model.

2.3 Convolutional neural nets

Feed-forward neural nets together with backpropagation algorithm have showed very useful for range of tasks and it has been even proven [9, 27] they can approximate any continuous function. However, they are not very good in recognizing objects presented visually. As every unit is connected to large amount of units in the previous layer (or all of them in fully-connected layers), the number of weights grows rapidly with the size of the problem and even more with the dimensionality. All these issues are becoming apparent even in image processing, which has only two dimensions. Convolutional neural nets (CNN) were introduced as a way to reduce the number of parameters involved, while exploiting the spatial constraints of the input.

CNN ideas took inspiration from neurobiology, more precisely the organisation of neurons in visual cortex of the cat. They were first used in the work of Homma [1], to process a temporal signal, and their design was later improved by LeCun et al. [36]. Different CNN architecture was proposed by Graupe [17] for decomposing one-dimensional EMG signals[4]. Convolutional nets can be also used in natural language processing [33] and analysis of three-dimensional data like videos [29] or volumetric data (e.g. 3D medical scans), but that is not as common as image processing.

Basic architecture of CNN can be described by the following process:

[3]Fusional language is a type of language distinguished by its tendency to overlay many morphemes to denote grammatical, syntactic, or semantic change.

[4]Electromyography (EMG) is an electrodiagnostic medicine technique for evaluating and recording the electrical activity produced by muscles.

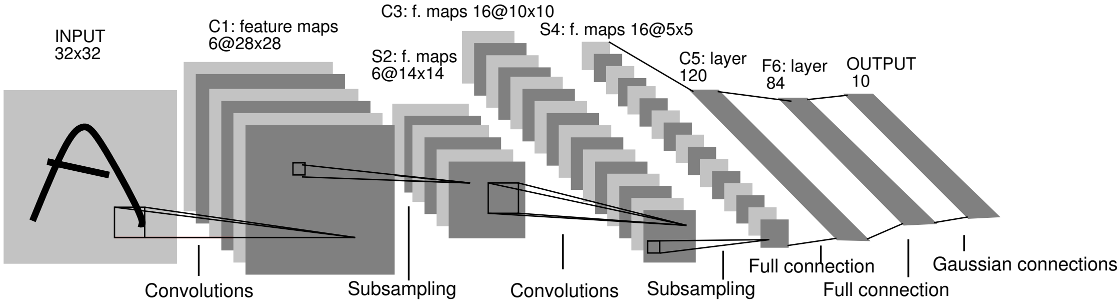

Figure 2.6: Architecture of famous CNN *LeNet-5*. [36]

1. Convolve several small filters on the input.
2. Subsample this space of filter activations.
3. Repeat steps 1 and 2 until you are left with a sufficiently high level features.
4. Use a standard feed-forward neural net to solve the task, using the features from step 3 as input.

Thus the CNN consist of alternating convolutional and subsampling layers, followed by fully connected feed-forward network. Diagram of the simple CNN architecture is on Figure 2.6. I will now go through the individual network segments and describe them.

Convolutional layer, which is most important and gave CNNs their name, is essentially the same as mathematical convolution used elsewhere. Here it means to apply a 'filter' over an input at all possible offsets. This filter - in image processing and computer vision called kernel - has a layer of connection weights with the same dimensionality as the input, but with much smaller size. Despite the fact that there is many connections in one convolution, which are even overlapping, the weights are tied together and only handful of parameters per filter need to be updated during training. Usually, several filters, ranging from 5 to 100, are applied to the input simultaneously in one layer. The main reason why it is possible to use this architecture is the ability to stack convolutions on top of each other to create more high-level from low-level features, while keeping the proportions of input.

As the output of the network usually do not have same dimensions of the input, ability to directly control size of the features is needed. In CNNs it is provided by subsampling, or in this version max pooling, layer. It is a simple operation that takes small non-overlapping grid of the input tensor and outputs the maximum value of each part. By putting this operation in between the convolutional layers, we can scale the current feature tensor and detect higher level features than without it.

Nowadays, most popular way to introduce nonlinearity to CNN is inserting rectifiers after convolutions, as they have excellent performance, surpassing any other unit [28, 43]. In the second, fully connected, part mix of linear units and ReLUs is commonly used, with applying dropout to these layers. Connection between convolutional and fully connected part is provided by a layer converting higher-dimensional output data from convolutions to a one-dimensional input vector.

CNNs are useful in applications, where data has a spatial structure, which is useful to capture in the model. Among these data belongs image processing and speech recognition. One of the first and most famous examples of convolutional neural net is LeNet[5] [36], which recognize handwritten digits from the MNIST database[6]. Figure 2.6 shows the architecture of LeNet, version called *LeNet-5*.

[5] Demos and examples of LeNet: http://yann.lecun.com/exdb/lenet/
[6] MNIST database website: http://yann.lecun.com/exdb/mnist/

Chapter 3

Image caption generation

Scene understanding is one of the fundamental, but also most difficult tasks of computer vision and ability to automatically generate text captions of an image or video can have a great impact on lives of many. However, it is much more complicated than simple classification or object recognition tasks, because the model also need to understand relations between the recognized objects and encode that relationship correctly in the caption.

In this chapter, I have done an overview of state-of-the-art approaches to the image captioning task and more closely describe latest studies, which are the basis of this work (section 3.1). Following section 3.2 cover popular datasets. Last section 3.3 covers evaluation procedures, which are most commonly used for this task.

3.1 Related Work

Two main approaches to image captioning were popular, until neural networks dominated the field. The first one used caption templates, which were filled by detected objects and relations. Second was based on retrieval of similar captions from database and modifying them to fit the current image. Question of similarity ranking has been addressed by many papers, which are based on the idea of joint embedding vector space for both images and captions [32], as it transforms estimation of similarity to a simple proximity measurement. Both approaches included a generalization step to remove information relevant only to current image, for example names.

These approaches were quite successful in describing images, but they are heavily hand-designed. Also their text-generation power is fixated on the database/embeddings and is not able to describe previously unseen compositions of objects. Over time these approaches fell out of favor, as methods leveraging the power of neural networks emerged. However, some of their ideas proved to be useful in the new environment and we can encounter them in recent works [14].

Many of the new methods, which use neural nets, are inspired by the success in training recurrent nets for machine translation. It is worth mentioning Sutskevers work [51], which studied general sequence to sequence mapping by converting an input sequence to the fixed length vector, which is then decoded to the output sequence. This encoder–decoder architecture is closely related to the autoencoders and work of Kalchbrenner and Blunsom [31], who were first to map the entire input sequence to vector.

The introduced encoder–decoder architecture is important to the captioning task, because image description problem can be interpreted as a translation from an image to a sentence. In this case, encoder part of the model is usually a convolutional neural net, as they are excellent in the image classification [52]. Decoder part is the similar to the one in machine translation models – an RNN or a type of LSTM, as the output for both tasks is essentially the same.

Following the encoder–decoder idea, current image captioning research is shifting towards models, which are trained end-to-end with some type of stochastic gradient descent (SGD) algorithm. The reasons for the shift can be simplicity and a lot less hand design than in other methods. Different type of the state-of-the-art models are based on proven pipeline of key-word detection, sentence generation, and ranking, which exploit the power of embedded neural networks, which specialize in single task. This approach is more closely described in section discussing article *From Captions to Visual Concepts and Back*.

The current field is consolidating, thanks to MS COCO Captioning Challenge[1] and dataset created for it. Simple public access to the necessary data makes model creation easier and the best researchers can compete directly against each other by using MS COCO evaluation server. In the rest of this section, I describe several works, which were submitted to the challenge in 2015 and had the best performance. MS COCO dataset is described, together with other datasets, in following section 3.2.

Show and Tell: A Neural Image Caption Generator

Show and Tell [54] is model created by Google researchers, which tied for the first place in MS COCO Captioning Challenge with the following model *From Captions to Visual Concepts*. The main idea of this work is to use recent advancements in machine translation and apply them for image captioning. Model uses encoder–decoder architecture, with CNN for the encoder part and RNN for the decoder part, as described earlier. Model is trained to maximize the likelihood $p(S|I)$ of producing a target sequence of words $S = \{S_1, S_2, ...\}$ given an input image I.

Used convolutional neural net has been pre-trained for an image classification task and last hidden layer of this network has been used as an input to the RNN decoder. The RNN part of the network is made of LSTM units based on following equations:

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1}) && (3.1)\\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1}) && (3.2)\\
\widetilde{C}_t &= \tanh(W_{xc}x_t + W_{hc}h_{t-1}) && (3.3)\\
C_t &= f_t \odot C_{t-1} + i_t \odot \widetilde{C}_t && (3.4)\\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1}) && (3.5)\\
h_t &= o_t \odot C_t && (3.6)
\end{aligned}
$$

Notation is same as in the chapter 2, σ is the standard logistic function defined in Equation (2.9) and the operation $\odot$ denotes the element-wise vector product. It is worth noticing the LSTM version used do not have "peephole connections". Several more changes were added – the second evaluation of hyperbolic tangens in Equation (3.6) is missing, as well as biases in all equations.

The language model is working on the word-level, part of the RNN is word embedding [39], which is trained together with the model. CNN, which is used to generate a configuration vector from the image, is connected to the RNN at the beginning as the first

[1] MS COCO Captioning Challenge: http://mscoco.org/dataset/#captions-challenge2015

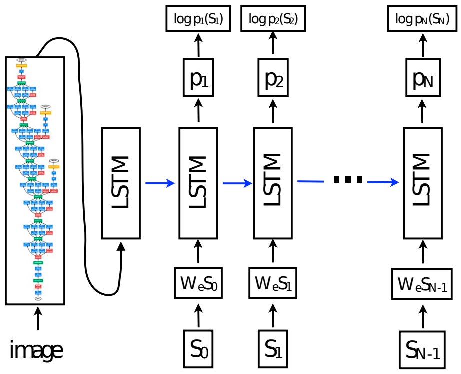

Figure 3.1: *Show and Tell* image captioning model. [54]

input before the generated sequence. Overall structure of the model is visualized on Figure 3.1 and can be represented by following equations:

$$x_{-1} = CNN(I) \tag{3.7}$$

$$x_t = W_e S_t \qquad t \in \{0 \dots N-1\} \tag{3.8}$$

$$p_{t+1} = LSTM(x_t) \qquad t \in \{0 \dots N-1\} \tag{3.9}$$

As the image and word encodings are used in the same way, model is effectively mapping both images and words into the same vector space. During the sequence input, special start word S_0 and stop word S_N designated to mark start and end of the sequence are used.

The CNN component of the model has been initialized to an ImageNet trained model, which helped quite a lot in terms of generalization. Word embeddings were left uninitialized (initialized randomly) as they did not observed significant gains while using large corpus. Dropout and ensebling used during training gave minor improvements. Model has been trained using SGD with fixed learning rate and no momentum. For the embeddings vector and the LSTM memory 512 dimensions were used. During the inference, beam search has been used to improve the results.

From Captions to Visual Concepts and Back

Predelat

This paper [14] took quite a different approach than a previous one, however both tied for the first place in the competition. This model is not end-to-end, it has three stages. First, it learns to extract nouns, verbs and adjectives from regions in the image. These words come from the vocabulary constructed by using 1000 most common words in the training captions. By running detector on the image regions, model is able to produce bag of bounding boxes. Each bounding box represent location of word in the image. Network pretrained on ImageNet is used for initialization.

Second, these extracted words guide a language model to generate text, which include these words. The maximum entropy (ME) language model estimates the probability of a word w_i conditioned on the preceding words, as well the words with high likelihood detections, yet to be mentioned. This encourages all the words to be used, while avoiding repetitions. A left-to-right beam search similar to [46] is used during generation. After extending each sentence with a set of likely words, the top N sentences are retained and the others pruned away. The process continues until a maximum sentence length L is reached.

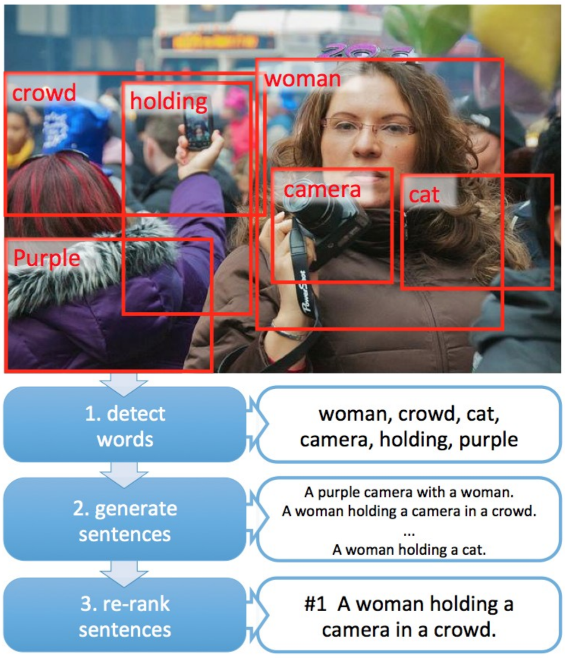

Figure 3.2: *From Captions to Visual Concepts* caption generation pipeline. [14]

Third, candidate captions are re-ranked using Minimum Error Rate Training [44] (MERT) and the best one is selected. MERT uses linear combination of features computed over the sentence, for example log-likelihood of the sequence or its length. One of the features is Deep Multimodal Similarity Model (DMSM) score, which measures similarity between images and text. The DMSM is model proposed in this paper, which learns two neural networks that map images and text fragments to a common vector representation. These vectors are used to compute the cosine similarity score, which is one of the features fed to MERT.

Direct comparison of this approach with the first one, presented in the paper description **??**, is in paper [10] by the same authors. They examine the issues of both approaches and achieve state-of-the-art performance by combining key aspects of RNN and ME methods.

Show, Attend and Tell: Neural Image Caption Generation with Visual Attention

Predelat

Work [57] of researchers from universities in Toronto and Montreal was inspired by recent work in machine translation and introduced an attention based model. Attention is one of the most interesting parts of the human visual system. Rather than compressing an entire image into a static representation, attention allows for salient features to dynamically come to the forefront as needed. Proposed model has encoder-decoder architecture. Encoder part use a convolutional neural network to extract set of feature/annotation vectors (not just one). Each of the vectors correspond to a part of image. Features from a lower convolutional layer are used to obtain a correspondence between them.

Decoder part is a LSTM network working of the word level, which generates, apart from the word of the output, a context vector - a dynamic representation of the relevant part of the image at time t. The paper explored two attention mechanisms computing the

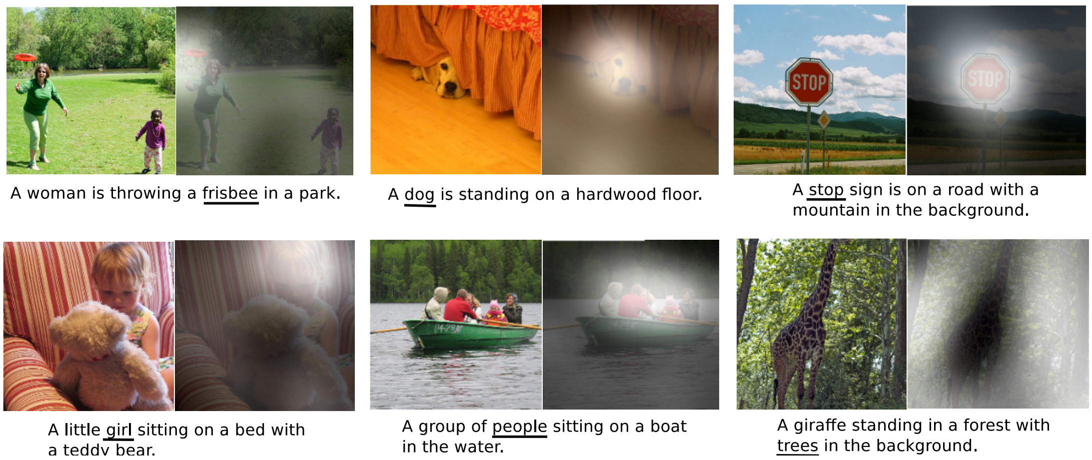

Figure 3.3: Examples of attending the correct object while generating the word. [57]

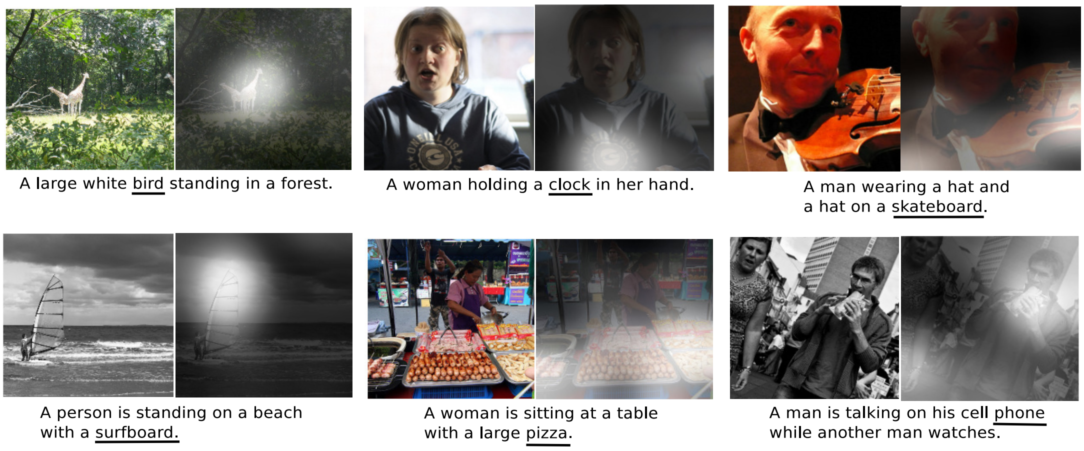

Figure 3.4: We can use attention to understand, what was the model looking at, while generating wrong caption. [57]

context vector from the annotation vectors. First is the stochastic "hard" mechanism, which interprets the values in the context vector as the probability that corresponding location is the right place to focus for producing the next word. Second is deterministic "soft" mechanism introduced in [2]. It gives the relative importance of the location by blending values for all annotation vectors together. This method is fully trainable by standard back-propagation methods.

Paper is also showing how we can gain insight and interpret the results of the model by visualising where and what the attention was focused on. Examples of the correct visualisations is on the figure 3.3 and the wrong ones on the figure 3.4. Visualisations show that model can attend even "non-object" regions. This adds an extra layer of interpretability to the output. The model learns alignments that correspond very strongly with human intuition. Especially in the examples of mistakes, we can understand why those mistakes were made.

CNN trained on ImageNet without finetuning was used for the decoder part. Model was trained with several algorithms and researchers found that for Flickr8k dataset, RMSProp worked best, while for Flickr30k and MSCOCO datasets, Adam [34] algorithm was used.

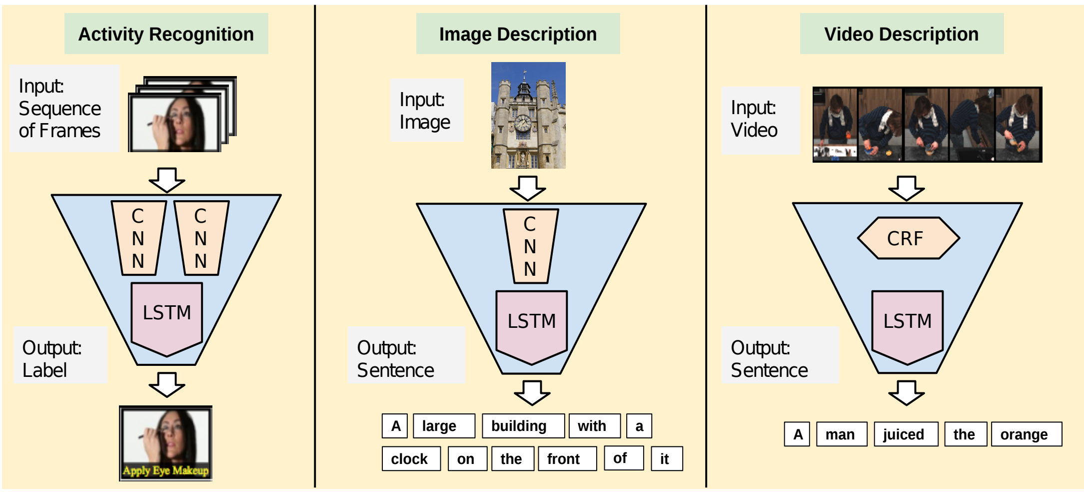

Figure 3.5: Task-specific instantiations of the LRCN model. [11]

Performance during training was also improved by creating minibatches of sentences with same length, which greatly improved convergence speed.

Long-term Recurrent Convolutional Networks for Visual Recognition and Description

Predelat

The research group from Berkeley in paper [11] presented long-term recurrent convolutional network (LRCN), which combines convolutional and long-range temporal layers for several tasks. It is possible to apply the introduced network to recognize activity performed on the video (sequential input $\mapsto$ fixed output), generate description of the image (fixed input $\mapsto$ sequential output) or describe video (sequential input $\mapsto$ sequential output).

Architecture of the proposed model is similar to the one from part **??**, only output of the CNN is fed to the LSTM in each time step. According to the task specification model can use separate convolutional networks, different for each time step, each with specific input or single CNN through all the time steps. Example is on the figure 3.5.

3.2 Datasets

Large amounts of data are necessary requirement in training deep neural nets like CNNs and RNNs, as well as sufficient computing power. Access to the machines and hardware suitable for training has been made in recent years extremely easy, with the rise of virtualization services. Obtaining enough data is different issue and it is currently the biggest problem. Especially, creation of image captioning datasets is quite complicated. As there is no automatized way to generate data, all the image descriptions have to be human-generated. This is one of the reasons, only few specialized datasets are created.

There are two main options how to get images and captions. First way is, by using user-generated data from an online service, most commonly Flickr[2]. However, these captions are not made specifically for the task and could be prone to error. Second option is to gather

[2] Flickr is a popular image and video hosting website and an online community. (https://www.flickr.com)

only images, again from Flickr or other online services, and create captions for direct use in the dataset manually. Amazon Mechanical Turk[3] is heavily used for this task.

The following table 1 lists the most popular datasets. All these datasets were created directly for the image captioning task, with captions generated through Amazon Mechanical Turk. Flickr8k [25], from 2013, was one of the first datasets created for this purpose. It has been later expanded into Flickr30k [58]. The biggest dataset is Microsoft Common Objects in Context (MS COCO) [6], created for the MS COCO captioning challenge. CIDEr [53] datasets PASCAL-50S, ABSTRACT-50S are youngest mentioned, designed specifically for evaluation with the CIDEr metric discussed in section 3.3.

Table 1: Image captioning datasets.

Name	**Images**	**Captions per image**	**Note**
MS COCO[4]	120 000	5	Images are divided - 80 000 for training and 40 000 for testing purposes.
Flickr30k[5]	31 783	5-6	An extension of Flickr8k dataset.
Flickr8k[6]	8 092	5	Focused on people or animals (mainly dogs) performing some specific action.
PASCAL-50S[7]	1 000	50	Built upon images from the UIUC Pascal Sentence Dataset.
ABSTRACT-50S[8]	500	50	Built upon images from the Abstract Scenes Dataset. No photos.

3.3 Evaluation

Recent progress in fields like machine translation, which are very similar to image captioning, caused spike of interest in evaluating regular text output accuracy. Although, sometimes it is not clear, if a description of an image is the best option available, some degree of assessment is possible. The best results can be obtained by asking live raters to score the usefulness of each description. Subjective scores can vary, but their average over many raters are usually quite accurate. However, this method consumes tremendous

[3] Amazon Mechanical Turk is crowdsourced Internet marketplace for tasks that computers are currently unable to do. (https://www.mturk.com)

[4] MS COCO project: http://mscoco.org/dataset/

[5] Flickr30k project: http://shannon.cs.illinois.edu/DenotationGraph/

[6] Flickr8k project: http://nlp.cs.illinois.edu/HockenmaierGroup/8k-pictures.html

[7] PASCAL-50S and ABSTRACT-50S: http://ramakrishnavedantam928.github.io/cider/

[8] See footnote 7.

amount of time and external raters are necessary in most cases. Like with data generation, tools like Amazon Mechanical Turk are used to great extent, but need for automated tools is evident.

3.3.1 Automated metrics

Assuming that one has access to human-generated captions, which is ground truth in our case, completely automated metrics are available. Even though all of them compute how alike are model descriptions to human-generated, different ratings are used by different metrics and even differences between used settings and implementations of one metric can invalidate the results. This raises the question, how can we compare results of different works, despite them using the "same" evaluation method. Microsoft group of researchers, team responsible for MS COCO, addresses this issue in [6]. They created an evaluation server[9], which has many automated metrics, with several configurations, including all mentioned here. This server should serve as a reference point for comparison of image captioning models.

Among the most popular metrics belong BLEU, METEOR, and CIDEr. The rest of this section is describing and discussing their properties. BLEU (Bilingual Evaluation Understudy) [45] has been the most commonly used metric, which was created in 2002 to evaluate quality of machine translated text. Scores are computed on individual segments, usually sentences. BLEU has high correlation with human judgments and is very popular, even for captioning tasks. However, it is becoming outdated, as according to this metric, automatic methods are now outperforming humans, which is senseless. Four different variations of BLEU are used in MS COCO evaluation server.

METEOR (Metric for Evaluation of Translation with Explicit Ordering) [35] is another metric for the evaluation of machine translation, slightly younger than BLEU, from 2007. Scoring generated translations is performed by aligning them to one or more reference translations. Metric was designed to fix some problems of the BLEU. It can also look for synonyms and perform stemming on input words.

Metric designed directly to caption evaluation – CIDEr (Consensus-based Image Description Evaluation) [53] was introduced in 2015. This is still a very new metric, but with growing popularity as it correlate well with human judgment. Main idea of this metric is improving quality of the metric with growing number of captions for the single image. This can be observed on datasets introduced with it (see section 3.2).

[9] MS COCO evaluation server: http://mscoco.org/dataset/#captions-upload.

Chapter 4

Model design

Jak vypada celkovy navrh modelu, velky diagram. Pak jednotlive sekce s tim jak vypadaji casti modelu, klidne zoom velkeho diagramu.

4.1 General architecture

4.1.1 Convolutional net

4.1.2 Language model

4.2 Learning algorithm and training

Algorithms and data.

As a result of the previous research, I propose a model for the image captioning, which has a encoder–decoder architecture. Image is encoded by a convolutional neural network into a vector, which is passed to a first recurrent layer. The decoder is created by several LSTM or GRU layers. Image vector will be saved as value in memory of the LSTM unit. Sentence will be generated on character level, without embeddings. Architecture is on the figure 4.1.

First question, this architecture is supposed to answer, is, whether model without the support of embeddings can compete with other contemporary solutions. Second is, how much will the model improve, if LSTM units are set with a large bias, as described in [30] and in part **??**.

Model will work with one-hot character level encoding, with two extra special characters `<start>` and `<stop>`, which respectively mark beginning and the end of sequence. Size of the first LSTM layer will be between $100 - 1000$ items, number similar to the image embedding models mentioned in [54, 14].

Training will be performed on MS COCO and Flickr datasets, mentioned in part 3.2. I would also like to explore Adam [34] and several other optimization algorithms.

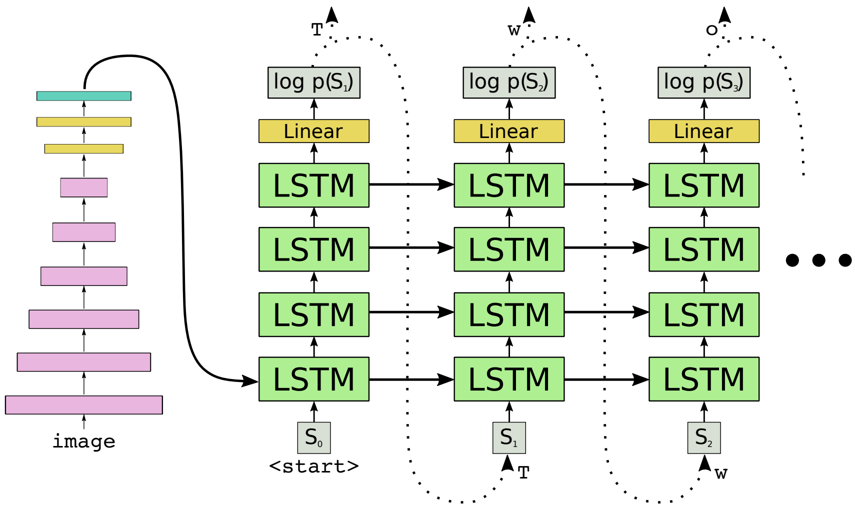

Figure 4.1: Architecture of the proposed model.

Chapter 5

Implementation

5.1 Tools

In this chapter I will describe tools and frameworks, which are used to implement deep learning models. *Torch* will be discussed separately in its own section as it was a tool I used for experimenting with language modeling - predicting next character in text, which is second part of this chapter.

One of the tools most academic researchers in deep learning rely on is *Theano*[1] [4], which is Python library that works with mathematical expressions and matrices. It is built upon *NumPy* to handle multidimensional arrays and compiles expressions before use for efficient computation. Theano can be quite intimidating and non-intuitive for some people, as it is focused on researchers and creating new architectures. For this reason, many packages and libraries has been created on top of Theano to simplify and streamline development of standard models. Among most popular are *Keras*[2], *Lasagne*[3], *Blocks*[4], which are open-source and available on GitHub, and *PyLearn2*[5].

Next Python library, independent of Theano, is *TensorFlow*[6], a tool from Google, released last year. It is used to process symbolic data flow graphs on many different types of machines, ranging from multiple GPU computers to smartphones. Interesting feature is ability to perform partial subgraph computation, which allows distributed training of the neural network. TensorBoard is a tool worth mentioning, as it provides visualisations of training and evaluation of the model, tool missing in most of the other libraries.

Python tools with the engine implemented in C/C++ are not the only ones. *Caffe*[7] is well-known and widely used library with API in C++. It performs very well in image classification with neural nets and can be used as a source of many pre-trained models hosted on the Model Zoo[8] site. It is also possible to use *Deeplearning4j*[9], which is deep-learning library written for Java and Scala, and a lot of others. Nowadays, libraries are introduced almost every month as this field is very live, which also means not everything is implemented in every framework and users need to follow news about their tools.

[1] Theano: http://deeplearning.net/software/theano/
[2] Keras: https://github.com/fchollet/keras
[3] Lasagne: https://lasagne.readthedocs.org/en/latest/
[4] Blocks: https://github.com/mila-udem/blocks
[5] PyLearn2: http://deeplearning.net/software/pylearn2/
[6] TensorFlow: https://www.tensorflow.org/
[7] Caffe: http://caffe.berkeleyvision.org/
[8] Caffe Model Zoo: https://github.com/BVLC/caffe/wiki/Model-Zoo
[9] Deeplearning4j: http://deeplearning4j.org/

5.1.1 Torch

Torch[10] is an open source scientific computing framework and machine learning library for the Lua. Underlying implementation is using extremely fast LuaJIT and C, but no need to code in C is required. Torch is not as popular in academic environment as Theano, but it is used by several large companies including Google & DeepMind, Facebook and IBM, which also contribute to the project. Apart from the companies, Torch has a large ecosystem of community-driven packages[11] with almost every tool needed for machine learning, computer vision and signal processing, and wide range of utilities. In the rest of this section I will describe fundamental Torch packages, which are necessary for my research.

The core package of Torch is *torch*, which is installed together with the library. It contains data structures for multi-dimensional tensors and operations over them. This is the most important part, as almost every package depends on them. Additionally, it provides many utilities for accessing files, serializing objects, processing command-line parameters and other useful utilities.

nn, nngraph

The base Torch provides necessary math structures, but the *nn* package allows simple creation of neural networks with a common `Module` interface. Module represents a layer of the network, which is the building block of the nets in Torch. Layers have `forward()` and `backward()` method and can be joined together by module composites `Sequential`, `Parallel` and `Concat`. These components allows creation of arbitrary graphs.

The *nn* package also contains loss functions, which are subclasses of `Criterion`. Classes `ClassNLLCriterion` and `CrossEntropyCriterion` contain common cross-entropy classification criterion. Other regression and embedding criterions are also available together with simple method to train the network with stochastic gradient descent.

Creating networks with complex graphs is quite complicated with the *nn*. To make it easier, *nngraph* package has been introduced. *nngraph* bundles *nn* modules into graph nodes, which are linked together by specifying inputs and outputs. Graphs can be visualized by `dot()` method and exported to vector graphics.

Both packages are sufficient and provide even advanced features like weight-sharing or weight-tying. They are mainly focused on feed-forward and convolutional networks. Creating RNNs is possible, however it is very complicated and labor-intensive.

rnn

Torch's *rnn* [38] package extends *nn* can be used to build recurrent neural nets, LSTM and GRU layers, and so on. The package handles the unrolling of a network and provides several options how to train a network. One of the ways is to use `backwardOnline()`, call `forward()` method repeatedly and then go `backward()` in the opposite order. Other option is to decorate model with `Sequencer` and feed the sequence to the network with only one `forward()` and one `backward()` call.

Package also provide module for implementing attention model [42]. It is worth mentioning that creating complex recurrent networks with the *nngraph* package might be difficult, as both packages are altering *nn* functionality and may collide.

[10] Torch: http://torch.ch/

[11] Wikipage with list of packages: https://github.com/torch/torch7/wiki/Cheatsheet

Predelat, pridat baliky.

Other packages

As the number of community packages is enormous, I will list just a few, which are commonly used and related to the topic of this work:

- *optim* — Optimization library with SGD, AdaGrad, RMSProp and more.
- *nninit* — Weight initialisation schemes for nn modules
- *image* — Routines to load/save and manipulate images as *torch* tensors.
- *word2vec* — Pre-trained word embeddings and the distance metric.
- *cunn*, *clnn* — CUDA and OpenCL backends for *nn* package.
- *loadcaffe* — Method for loading Caffe models in Torch.

5.2 File structure and workflow

Nejaky popis jake soubory jsem udelal, co delaji, jak je spoustet pri trenovani a pri samplovani... odstavec o tom na jakem hardwaru se trenovalo, jak dlouho to trvalo atd.

Pretraining language model

Combined model

Chapter 6

Experiments and results

Nejaka poznamka o tom na jakem hardwaru se trenovalo, jake melo gpu a stroj parametry.

6.1 Language model initialization variations

6.2 Evaluation

6.3 Analysis of results

Chapter 7

Conclusion

Predelat

Image captioning is problem more difficult than simple classification of images. Nowadays, deep neural networks are dominating the field almost exclusively. In this work I explained several features of neural networks, which are necessary for creation of image captioning models, and created an overview of state-of-the-art approaches to this problem. Deep learning is research area with a great need of data, therefore I also listed several biggest and most commonly used datasets and evaluation methods.

Character level prediction models were created as a building block for creating my own image captioning model, based on the information compiled. I proposed an architecture of the model based on several other approaches, with interest in character level language models instead of word embedding. I explore whether this simple architecture can compete with contemporary solutions focused on word level language models.

In several following months I am planning to work on the details of the convolutional network architecture, implementation and training of the proposed model. Project should be finished by May 2016.

Bibliography

[1] Les E. Atlas, Toshiteru Homma, and Robert J. Marks II. An Artificial Neural Network for Spatio-Temporal Bipolar Patterns: Application to Phoneme Classification. In *Neural Information Processing Systems*, pages 31–40. American Institute of Physics, 1988.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *CoRR*, abs/1409.0473, 2014.

[3] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[4] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A CPU and GPU Math Compiler in Python . In *Proceedings of the 9th Python in Science Conference*, pages 3 – 10, 2010.

[5] Arthur Bryson and Yu-Chi Ho. *Applied Optimal Control: Optimization, Estimation and Control*. Halsted Press book'. Taylor & Francis, 1975.

[6] Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO Captions: Data Collection and Evaluation Server. *CoRR*, abs/1504.00325, 2015.

[7] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *CoRR*, abs/1406.1078, 2014.

[8] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *CoRR*, abs/1412.3555, 2014.

[9] G. Cybenko. Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

[10] Jacob Devlin, Hao Cheng, Hao Fang, Saurabh Gupta, Li Deng, Xiaodong He, Geoffrey Zweig, and Margaret Mitchell. Language Models for Image Captioning: The Quirks and What Works. *CoRR*, abs/1505.01809, 2015.

[11] Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term Recurrent Convolutional Networks for Visual Recognition and Description. *CoRR*, abs/1411.4389, 2014.

[12] Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. Incorporating second-order functional knowledge for better option pricing. *Advances in neural information processing systems*, pages 472–478, 2001.

[13] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[14] Hao Fang, Saurabh Gupta, Forrest N. Iandola, Rupesh K. Srivastava, Li Deng, Piotr Dollár, Jianfeng Gao, Xiaodong He, Margaret Mitchell, John C. Platt, C. Lawrence Zitnick, and Geoffrey Zweig. From Captions to Visual Concepts and Back. *CoRR*, abs/1411.4952, 2014.

[15] Felix A. Gers and Jürgen Schmidhuber. Recurrent Nets that Time and Count. In *IJCNN (3)*, pages 189–194, 2000.

[16] Felix A. Gers, Jürgen Schmidhuber, and Fred A. Cummins. Learning to Forget: Continual Prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 2000.

[17] D. Graupe, Ruey wen Liu, and George S. Moschytz. Applications of neural networks to medical signal processing. In *Decision and Control, 1988., Proceedings of the 27th IEEE Conference on*, pages 343–347 vol.1, December 1988.

[18] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. *CoRR*, abs/1303.5778, 2013.

[19] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing Machines. *CoRR*, abs/1410.5401, 2014.

[20] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A Search Space Odyssey. *CoRR*, abs/1503.04069, 2015.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *CoRR*, abs/1512.03385, 2015.

[22] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[23] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. *Master's thesis, Technische Universität München*, 1991.

[24] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997.

[25] Micah Hodosh, Peter Young, and Julia Hockenmaier. Framing Image Description as a Ranking Task: Data, Models and Evaluation Metrics. *Journal of Artificial Intelligence Research*, 47:853–899, 2013.

[26] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the national academy of sciences*, volume 79, pages 2554–2558. National Acad Sciences, 1982.

[27] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4:251–257, 1991.

[28] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the Best Multi-Stage Architecture for Object Recognition? In *Proceedings of the International Conference on Computer Vision (ICCV'09)*. IEEE, 2009.

[29] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3D Convolutional Neural Networks for Human Action Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, 2013.

[30] Rafal Józefowicz, Wojciech Zaremba, and Ilya Sutskever. An Empirical Exploration of Recurrent Network Architectures. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 2342–2350, 2015.

[31] Nal Kalchbrenner and Phil Blunsom. Recurrent Continuous Translation Models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709. Association for Computational Linguistics, 2013.

[32] Andrej Karpathy and Fei-Fei Li. Deep Visual-Semantic Alignments for Generating Image Descriptions. *CoRR*, abs/1412.2306, 2014.

[33] Yoon Kim. Convolutional Neural Networks for Sentence Classification. *CoRR*, abs/1408.5882, 2014.

[34] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *CoRR*, abs/1412.6980, 2014.

[35] Alon Lavie and Abhaya Agarwal. METEOR: An Automatic Metric for MT Evaluation with High Levels of Correlation with Human Judgments. In *Proceedings of the Second Workshop on Statistical Machine Translation*, StatMT '07, pages 228–231, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.

[36] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-Based Learning Applied to Document Recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, November 1998.

[37] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[38] Nicholas Léonard, Sagar Waghmare, Yang Wang, and Jin-Hwa Kim. rnn : Recurrent Library for Torch. *CoRR*, abs/1511.07889, 2015.

[39] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *CoRR*, abs/1301.3781, 2013.

[40] Tomas Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, September 26-30, 2010*, pages 1045–1048, 2010.

[41] Tomas Mikolov, Wen tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2013.

[42] Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. Recurrent Models of Visual Attention. *CoRR*, abs/1406.6247, 2014.

[43] Vinod Nair and Geoffrey E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 807–814, 2010.

[44] Franz Josef Och. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, volume 1, pages 160–167. Association for Computational Linguistics, 2003.

[45] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[46] Adwait Ratnaparkhi. Trainable Methods for Surface Natural Language Generation. *CoRR*, cs.CL/0006028, 2000.

[47] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[48] P. Smolensky. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Information Processing in Dynamical Systems: Foundations of Harmony Theory, pages 194–281. MIT Press, Cambridge, MA, USA, 1986.

[49] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[50] Guo-Zheng Sun, C. Lee Giles, and H. H. Chen. The Neural Network Pushdown Automaton: Architecture, Dynamics and Training. In *Adaptive Processing of Sequences and Data Structures*, pages 296–345, London, 1998. Springer-Verlag.

[51] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to Sequence Learning with Neural Networks. *CoRR*, abs/1409.3215, 2014.

[52] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. *CoRR*, abs/1409.4842, 2014.

[53] Ramakrishna Vedantam, C. Lawrence Zitnick, and Devi Parikh. CIDEr: Consensus-Based Image Description Evaluation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[54] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and Tell: A Neural Image Caption Generator. *CoRR*, abs/1411.4555, 2014.

[55] P.J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* Harvard University Press, 1974.

[56] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical Evaluation of Rectified Activations in Convolutional Network. *CoRR*, abs/1505.00853, 2015.

[57] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. *CoRR*, abs/1502.03044, 2015.

[58] Peter Young, Alice Lai, Micah Hodosh, and Julia Hockenmaier. From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. *Transactions of the Association for Computational Linguistics*, 2:67–78, 2014.

[59] Matthew D Zeiler, Marc'Aurelio Ranzato, Rajat Monga, Min Mao, Kun Yang, Quoc Viet Le, Patrick Nguyen, Alan Senior, Vincent Vanhoucke, Jeffrey Dean, et al. On rectified linear units for speech processing. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 3517–3521. IEEE, 2013.

List of Abbreviations

BLEU Bilingual Evaluation Understudy.

CIDEr Consensus-based Image Description Evaluation.

CNN Convolutional neural network.

GRU Gated Recurrent Unit – type of layer in recurrent nets.

LSTM Long Short-Term Memory – type of layer in recurrent nets.

ME Maximum entropy.

MERT Minimum Error Rate Training.

METEOR Metric for Evaluation of Translation with Explicit Ordering.

MS COCO Microsoft Common Objects in Context – computer vision dataset and popular challenge.

ReLU Rectified Linear Unit – type of layer in convolutional nets.

RNN Recurrent neural network.

SGD Stochastic Gradient Descent.

Appendices

List of Appendices

A CD Content **38**

Appendix A

CD Content