Image Captioning using YOLO's output as Input to Encoder-Decoder LSTM

A DISSERTATION

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE AWARD OF THE DEGREE

OF

MASTER OF TECHNOLOGY

IN

SIGNAL PROCESSING & DIGITAL DESIGN

Submitted by:

Navneet Kumar

2K16/SPD/10

Under the supervision of

Shri Rajesh Birok

(Associate Professor)

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

MAY, 2018

CANDIDATE'S DECLARATION

I, (Navneet Kumar), 2K16/SPD/10 of M.Tech (Signal Processing & Digital Design), hereby declare that the project dissertation titled "**Image Captioning using YOLO's output as Input to Encoder-Decoder LSTM**" which is submitted by me to the department of Electronics & Communication, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology, is original & not copied from any source without paper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: Delhi Navneet Kumar

Date: (2K16/SPD/10)

CERTIFICATE

I hereby certify that the Project Dissertation titled "**Image Captioning using YOLO's output as Input to Encoder-Decoder LSTM**" which is submitted by Navneet Kumar, 2K16/SPD/10, Department of Electronics & Communication, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology, is a record of the project work carried out by the student under my supervision. To the best of my knowledge this work has not been submitted in part or full for any degree or diploma to this university or elsewhere.

Place: Delhi

Date:

RAJESH BIROK
(SUPERVISOR)
Associate Professor
Department of ECE

Abstract

The task of caption generation for image has recently received considerable attention. In this thesis we will see how we can make computers to look at an image and output a description for the same. This process has many potential applications in real life. A noteworthy one would be to save the captions of an image so that it can be retrieved easily at a later stage just on the basis of this description. With few modifications this system can also assist visually-impaired persons with their daily chores.

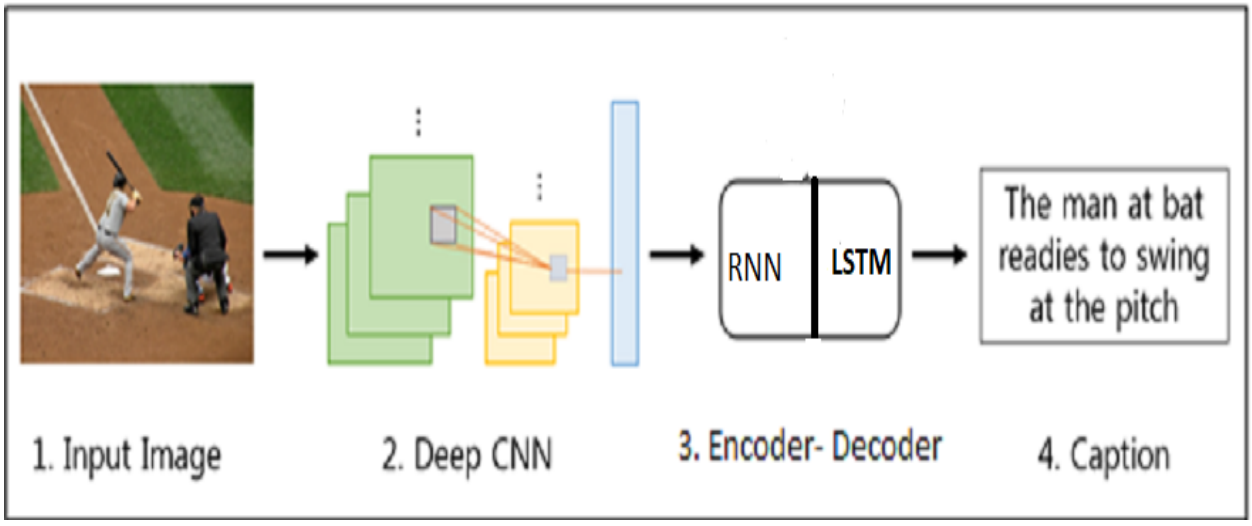
The task of caption generation is straightforward – Given an input image our algorithm is expected to describe what is there in the image. By description we mean that the system will tell us about the objects present in the image, and the tasks that are being performed by the objects. Tasks like these are trivial for humans, but non-trivial for computers. Thanks to advancement in deep learning, computers are now reaching human level performance.

This thesis work introduces a generic end-to-end trainable Convolutional Neural Network (CNN) -Recurrent Neural Network (RNN) Fusion-based technique to solve the problem of image captioning. In particular, we feed an image into a CNN and the output of CNN then gets fed to an Encoder-Decoder. The task of CNN is to output set of objects and their location. Encoder-Decoder network takes the output of CNN as input and feed that into its Encoder. The Encoder uses a two-stream RNN to encode the information coming from CNN, the coded information then gets passed to decoder. Decoder uses a standard LSTM neural network to generate text.

Standard MS-COCO captioning task dataset is used for this task.

Keywords : Convolutional Neural Network (CNN) , Recurrent Neural Network (RNN), Encoder-Decoder.

Visual Abstract



ACKNOWLEDGEMENT

I owe my gratitude to all the people who have helped me in this dissertation work & who have made my postgraduate college experience one of the most special periods of my life.

Firstly, I would like to express my deepest gratitude to my supervisor Shri Rajesh Birok, Associate Professor (ECE) for his invaluable support, guidance, motivation & encouragement throughout the period during which this work was carried out. I am deeply grateful to Dr. S. Indu, H.O.D. (Department. of ECE) for their support & encouragement in carrying out this project.

I also wish to express my heart full thanks to all faculties at Department of Electronics & Communication Engineering of Delhi Technological University for their goodwill & support that helped me a lot in successful completion of this project.

Finally, I want to thank my parents, family & friends for always believing in my abilities & showering their invaluable love & support.

Navneet Kumar

(2K16/SPD/10)

Chapter 1

1.1 Introduction

The famous English adage states “A picture is worth a thousand words”. Translated into technical terms, this is meant to convey that there is a lot the viewer can learn or infer from a single still image and that enumerating all the information encoded in an image can take up to even a thousand words. This is illustrated in our extensive use of images in all forms of communication, from scientific journals to Twitter chats. Humans are very good at processing images and videos and gathering all this encoded information, but the computers still struggle to make sense of the simplest ones. One could say it is still easier for computers to store, parse, search and even understand a thousand words than a single image. The use of multimedia on the internet has grown to staggering levels in the recent years, due to easy access to cameras in smartphones. For example about 95 million photos are uploaded to Instagram every day [2] and about 400 hours of video is uploaded to YouTube every minute [1]. Rapidly growing amount of visual data being created due to this phenomenon presents both an enormous challenge and an opportunity to build smarter computer algorithms to understand and summarize the data. Such algorithms could help us index and search this visual data better. An algorithm which can learn to recognize and describe different objects and their relationships in an image or a video would be an essential building block of a general artificial intelligence (AI) system. Hence, automatic understanding of visual media is an interesting and important problem in many aspects of computer vision and AI. An essential research topic at the heart of machine understanding of visual media is automatic captioning of images and videos. This involves designing an algorithm which takes the image or the video as input and generates a natural language caption succinctly describing the content of the media. Effectively solving the above problem requires the machine to be able to identify the salient objects in the image or video, recognize their attributes,extract the relationships between these objects and also to correctly recognize the scene. This machine also needs to be able to use the extracted information to generate a natural language caption summarizing the essence of it. Since the caption generation requires both visual feature extraction and natural language generation modules, it is also a good proxy task to measure the progress in both these domains.

Until recently, the task of reliably identifying even a single object in an image across diverse and large-scale datasets was hard. This changed dramatically with the availability of large-scale annotated data such as the ImageNet dataset [12], and the application of deep learning techniques, specifically convolutional neural networks (CNN). For example, in the image classification task of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [52], which involves classifying images to one of thousand object classes, the accuracy has improved from 71.8% to 95.06%, surpassing the human performance on the same task. It has also been discovered that image classification networks which are trained on the large ImageNet dataset, also generalize very well and can be used as generic image feature extraction for different tasks [74]. This has led to successful application of such deep networks to various other tasks in computer vision including the task of image captioning. In this thesis we will examine the task of automatic image and video captioning and discuss algorithms utilizing tools from deep learning to solve this task. Much of the discussion presented in this thesis applies to both image and video captioning problems. In such cases, in the interest of conciseness, we use the term “visual captioning” to refer to both of these problems.

In general, generating visually descriptive language can be useful for various tasks such as human-machine communication, accessibility, image retrieval, and search. However this task is still challenging and it depends on developing both a robust visual recognition model, and a reliable language generation model. In this paper, we instead tackle a task of describing object

Layouts where the categories for the objects in an input scene and their corresponding locations are known. Object layouts are commonly used for story-boarding, sketching, and computer graphics applications. Additionally, using our object layout captioning model on the outputs of an object detector we are also able to improve image captioning models. Object layouts contain rich semantic information, however they also abstract away several other visual cues such as color, texture, and appearance, thus introducing a different set of challenges than those found in traditional image captioning.

We propose OBJ2TEXT, a sequence-to-sequence model that encodes object layouts using an LSTM network (Hochreiter and Schmidhuber,1997), and decodes natural language descriptions using an LSTM-based neural language model. Natural language generation systems usually consist of two steps: content planning, and surface realization. The first step decides on the content to be included in the generated text, and the second step connects the concepts using structural language properties. In our proposed model, OBJ2TEXT, content planning is performed by the encoder, and surface realization is performed by the decoder. Our model is trained in the standard MS-COCO dataset (Lin et al., 2014), which includes both object annotations for the task of object detection, and textual descriptions for the task of image captioning. While most previous research has been devoted to any one of these two tasks, our thesis presents, to our knowledge, the first approach for learning mappings between object annotations and textual descriptions. Using several lesioned versions of the proposed model we explored the effect of object counts and locations in the quality and accuracy of the generated natural language descriptions.

Generating visually descriptive language requires beyond syntax, and semantics; an understanding of the physical word. We also take inspiration from recent work by Schmaltz et al. (2016) where the goal was to reconstruct a sentence from a bag-of-words (BOW) representation using a simple surface-level language model based on an encoder-decoder sequence-to-sequence architecture.

In contrast to this previous approach, our model is grounded on visual data, and its corresponding spatial information, so it goes beyond word reordering. Also relevant to our work is Yao et al. (2016a) which previously explored the task of oracle image captioning by providing a language generation model with a list of manually defined visual concepts known to be present in the image. In addition, our model is able to leverage both quantity and spatial information as additional cues associated with each object/concept, thus allowing it to learn about verbosity, and spatial relations in a supervised fashion.

In summary, our contributions are as follows:

1. We demonstrate that despite encoding object layouts as a sequence using an LSTM, our model can still effectively capture spatial information for the captioning task. We perform ablation studies to measure the individual impact of object counts, and locations.
2. We show that a model relying only on object annotations as opposed to pixel data, performs competitively in image captioning despite the ambiguity of the setup for this task.
3. We show that more accurate and comprehensive descriptions can be generated on the image captioning task by combining our OBJ2TEXT model using the outputs of a state-of-the-art object detector with a standard image captioning approach.

Caption: *a giraffe walking through a patch of high dried out grass.*

1.2 Problem Statement

We evaluate OBJ2TEXT in the task of object layout captioning, and image captioning. In the first task, the input is an object layout that takes the form of a set of object categories and bounding box pairs; $< o, l > = \{< o(i), l(i) >\}$, and the output is natural language. This task resembles the second task of image captioning except that the input is an object layout instead of a standard raster image represented as a pixel array. We experiment in the MS-COCO dataset for both tasks. For the first task, object layouts are derived from ground-truth bounding box annotations, and in the second task object layouts are obtained using the outputs of an object detector over the input image.

1.3 Related Work

Our work is related to previous works that used clipart scenes for visually-grounded tasks including sentence interpretation (Zitnick and Parikh, 2013; Zitnick et al., 2013), and predicting object dynamics (Fouhey and Zitnick, 2014). The cited advantage of abstract scene representations such as the ones provided by the clipart scenes dataset proposed in (Zitnick and Parikh, 2013) is their ability to separate the complexity of pattern recognition from semantic visual representation. Abstract scene representations also maintain common-sense knowledge about the world. The works of Vedantam et al. (2015b); Eysenbach et al. (2016) proposed methods to learn common-sense knowledge from clipart scenes, while the method of Yatskar et al. (2016), similar to our work, leverages object annotations for natural images. Understanding abstract scenes has demonstrated to be a useful capability for both language and vision tasks and our work is another step in this direction. Our work is also related to other language generation tasks such as image and video captioning (Farhadi et al., 2010; Ordonez et

al., 2011; Mason and Charniak, 2014; Ordonez et al., 2015; Xu et al., 2015; Donahue et al., 2015; Mao et al., 2015; Fang et al., 2015). This problem is interesting because it combines two challenging but perhaps complementary tasks: visual recognition, and generating coherent language. Fueled by recent advances in training deep neural networks (Krizhevsky et al., 2012) and the availability of large annotated datasets with images and captions such as the MS-COCO dataset (Lin et al., 2014), recent methods on this task perform end-to-end learning from pixels to text. Most recent approaches use a variation of an encoder-decoder model where a convolutional neural network (CNN) extracts visual features from the input image (encoder), and passes its outputs to a recurrent neural network (RNN) that generates a caption as a sequence of words (decoder) (Karpathy and Fei-Fei, 2015; Vinyals et al., 2015). However, the MS-COCO dataset, containing object annotations, is also a popular benchmark in computer vision for the task of object detection, where the objective is to go from pixels to a collection of object locations. In this paper, we instead frame our problem as going from a collection of object categories and locations (object layouts) to image captions. This requires proposing a novel encoding approach to encode these object layouts instead of pixels, and allows for analyzing the image captioning task from a different perspective. Several other recent works use a similar sequence-to-sequence approach to generate text from source code input (Iyer et al., 2016), or to translate text from one language to another (Bahdanau et al.,2015).There have also been a few previous works explicitly analyzing the role of spatial and geometric relations between objects for vision and language related tasks. The work of Elliott and Keller (2013) manually defined a dictionary of object-object relations based on geometric cues. The work of Ramisa et al. (2015) is focused on predicting preposition given two entities and their locations in an image. Previous works of Plummer et al. (2015) and Rohrbach et al. (2016) showed that switching from classification-

based CNN network to detection-based Fast RCNN network improves performance for phrase localization. The work of Hu et al. (2016) showed that encoding

image regions with spatial information is crucial for natural language object retrieval as the task explicitly asks for locations of target objects. Unlike these previous efforts, our model is trained end-to-end for the language generation task, and takes as input a holistic view of the scene layout, potentially learning higher order relations between objects.

1.4 Structure of the Thesis

The rest of the thesis is organized as follows. In Chapter 2, a discussion on the background literature related to the building blocks of caption generation models, i.e. visual feature extraction and language modeling, is presented. Here we will also review the several related works on visual captioning and discuss the datasets available to train such models. The details of our baseline caption generation model and the automatic metrics used to evaluate a captioning system are discussed in Chapter 3. Chapter 4 presents some extensions to the image and video features compared to the ones used in the baseline model. Chapter 5 discusses several extensions to the baseline language model and presents some ensembling techniques to combine multiple language models. Chapter 6 contains results from several experiments to determine the best configurations for our image and video captioning systems and provide comparisons to a few other state-of-the-art models from the literature. In Chapter 7, some shortcomings of our visual captioning systems are identified and a few interesting problems to explore to address these issues are discussed. The thesis is concluded in Chapter 9.

Chapter 2

Background: Vision & Language

In this chapter we will review some background literature exploring different facets of integrating visual data and their natural language annotations. This includes learning good representations for images and videos, different kinds of generative language models, techniques which attempt to rank similarity between visual data and language annotations, and finally some recent advances in captioning images and videos. Additionally, we will review some datasets available for training such captioning models

2.1 Visual Features

Visual media, be it image or video, are inherently very high-dimensional data. This large dimensionality poses a challenge for machine learning systems trying to extract higher-level semantic information directly from such visual inputs, as in case of captioning. To address this, images and videos have traditionally been represented with smaller feature vectors which attempt to encode the most important information present in them, while ignoring redundancies. This feature extraction step is very important in any image understanding pipeline as it usually serves as input to the subsequent modules and hence can be a major bottleneck to the performance of the entire system. Therefore, we will now review some feature extraction techniques for images and videos and identify the best performing ones, which will be used in designing the captioning system later.

2.1.1 Image Features

Traditionally, tasks such as object recognition have relied on using handcrafted features to represent images. Recently, however, deep Convolutional Neural Networks (CNN), which learn to extract features necessary for the task entirely from the data, have become a popular choice for image feature extraction. This was triggered by the spectacular improvement in image classification accuracy seen on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012, with the first use of CNNs in this competition. In this challenge, involving classifying the input images to one of thousand classes, the submission by Krizhevsky et al. [34] using a deep CNN outperformed all the others by a large margin. This set off further exploration into CNN architectures and has driven up the performance in the ImageNet classification task to even surpass the human classification accuracies [20]. More interestingly, the deep CNNs trained on the large ImageNet dataset for the classification task have been shown to generalize very well to other datasets and tasks as well. In [74], it is shown that the weights learned by the CNNs pre-trained on the ImageNet dataset are good initializers for other tasks as well. That is, if we use the weights from CNNs pre-trained on ImageNet to initialize the networks before training it on other datasets and tasks, we can learn much better models than just using random initialization. Alternatively, using activations from some higher layer of an ImageNet pretrained CNN as off-the-shelf image features has also been shown to produce state-of-the-art results [15, 54, 56, 57] in several datasets and tasks, including object detection, scene recognition, image and video captioning, etc. We will follow this second approach, i.e. use activations from CNNs pre-trained on ImageNet as feature input to our captioning model, without any fine-tuning of the CNNs for this task. GoogLeNet [59] and VGG [58] architectures, which won the different categories of

ILSVRC 2014 competition, have been popular models for such feature extraction in the community with the ready availability of the code and pre-trained models.

2.1.2 Video Features

Unlike in the case of images, where the convolutional neural network (CNN) image features have become the de facto standard features for many image understanding related tasks, no single video feature extraction method has achieved the best performance across tasks and datasets. Dense trajectories [66] and Improved dense trajectories [67] have been popular video feature extraction methods for the task of action recognition. In these methods, interest points are densely sampled in an initial frame and then tracked across frames, to form trajectories. Furthermore, a set of local descriptors are extracted around the trajectory coordinates to obtain a rich representation of the trajectory. These trajectories can encode motion patterns from a variety of sources, including actions from agents in the video, camera motion, etc. Following the success of the deep CNN models on static images, there have been attempts to train 3-D CNNs which operate directly on video segments [25, 31, 61]. However, these models need a lot of training data and are usually pre-trained on some large action recognition dataset, e.g. the Sports-1M dataset [31]. All of the above features encode action-related information very well, but fail to capture information about the identity of the objects in the video. The task of caption generation also requires us to describe the objects seen in the video and their attributes, in addition to recognizing actions. This can be addressed by extracting features from individual frames [64] or keyframes [55] using CNNs pre-trained on the ImageNet dataset. In this work we explore both

these paradigms of video feature extraction, namely hand-crafted trajectory features and deep video features, for the task of automatic visual captioning.

2.2 Natural Language Modeling

Generative language models are widely used in various tasks including speech recognition and synthesis, document analysis, dialog systems, etc. Our task of caption generation also involves learning a conditional generative language model which can generate captions given the input visual features. For this purpose, we will now discuss a few different language modeling approaches, and evaluate their suitability for our task. The simple n-gram language models, which are based on counting co-occurrence statistics of sequence of n words, are surprisingly good baselines for a lot of language Modeling tasks. However, they are constrained to generate sentences only by using n-grams they have seen in the training set. The maximum entropy language model (ME-LM) [5] overcomes this problem by using the principle of maximum entropy while learning the model. The principle dictates that among all the probabilistic models which satisfy the constraints of the training data, one should pick the model which is the most uniform. This allows the model to share some probability for unseen n-grams as well. Both the above models suffer from using a short context of previous words when predicting the next word, limited by the n-gram size, which can lead to longer generated sentences being incoherent. Alternatively, one could use recurrent neural networks as generative language models as proposed in [41]. Recurrent neural networks have the benefit of having access to theoretically infinite context through their hidden state. Additionally, these models do not need pre-defined language features, unlike in in case ME-LM , and can learn the necessary

word representations from the data. Indeed, such recurrent network based language models are quite popular in the machine translation task [3].

2.3 Intermediate Problem: Multi-Modal Embeddings

A precursor to the problem of caption generation from input visual features, is the problem of learning to map both visual features and the corresponding natural language labels into a common semantic space. This was posed as a solution to automatically annotate images in [68]. Here, using a simple bag-of-visual-words image representation, both the image and words are projected into a common space, with the training objective of ranking correctly the matching image and annotation pairs. In [18], using the CNN features as the image representation and the word vectors from word2vec [42] embeddings as the word representation, a linear embedding is learned to map the image vectors onto word vectors corresponding to the labels associated with the image. This allows the model to do "zero-shot" recognition of image classes it has not seen before, by finding the nearest label to the embedded image feature. A much simpler approach is used in [43], where the mapping to semantic embedding space is done by a convex combinations of c word vectors associated with the top c classes identified in the image. This does away with the need to learn the embedding, while still achieving impressive results in zero-shot classification on the ImageNet dataset. The methods discussed above forms the basis for the way visual features are used in captioning literature. In many early works, both image features and word vectors are input to the Long-Short Term Memory (LSTM) language model using the same input matrix, forcing the model to learn a joint embedding for them.

2.4 Approaches to Visual Captioning

Visual captioning techniques include a wide range of methods and models. They can be broadly categorized into two groups: ones generating captions by retrieving from a database [17, 24, 30], and ones using natural language generation techniques to produce captions [16, 35, 36, 65]. While the retrieval based methods tend to be semantically more accurate as they do not need to learn grammatical rules to generate a caption, the captions they produce are strictly restricted to the caption database, and thus will not work well on unseen data. In contrast, although generative models can learn to create novel captions and even perform reasonably well on unseen data, they tend to have poorer semantic accuracy and details. Early work on captioning images presented in [17] was retrieval based, wherein a similarity score between sentences and images is computed and is used to retrieve the best matching caption to an input image. This method relied on few hand-engineered image features and dependency-parsing-based features for sentences. In [24], authors pose image description as a ranking problem involving correctly ranking a set of captions by their relevance to the input image. They argue that the ability to rank captions correctly is a good measure of semantic image understanding, with the added benefit that it is much easier to automatically evaluate such ranked lists than to evaluate generated novel captions. Such retrieval-based system is further enhanced by the use of deep image features and word embeddings in [30]. One of the early models to successfully generate novel image captions was described [35], albeit relying on pre-defined sentence templates to generate captions. In [36], this template based language model is replaced with a n-gram based one learned from large-scale natural language data collected from the web. Following the successful use of recurrent network–based language models on tasks such as automatic speech recognition [41] and machine

translation [3], this approach was quickly adapted to the image captioning literature as well [14, 29, 65]. All these methods consist of two-stage encoder–decoder models, with the encoder being the image feature extraction module and the decoder being the recurrent language model. One major advantage of this approach is that it allows end-to-end training of the entire system. In the case of [29], CNN-based image features and a simple recurrent neural network (RNN) based language model are used as the encoder and the decoder, respectively. The authors also propose techniques to align different parts of a caption to different regions in the image. Similarly in [65], authors also use CNN image features, but an LSTM-based network is used for the language model. In their method, the image features are fed to the LSTM only at the beginning of the recursion, in order to prevent the network from overfitting. Starting from a similar CNN+LSTM based pipeline,[14] proposes a more general framework which can generate captions for both images and videos. In the case of videos, they replace the single image CNN feature with a sequence of conditional random field (CRF) video features and keep the language model configuration identical. As opposed to such end-to-end learning systems, [16] takes a modular approach. They first train a set of object or concept detectors using multiple instance learning [40]. Then, a maximum entropy language model takes these detector outputs as input to generate the candidate sentence. This concept detector–based approach has been applied also to video captioning in [51], where the authors train a “visual label” detector on the LSMDC dataset and use it as an input to an LSTM language model. In [72], instead of using a single image feature vector from a fully connected CNN layer, multiple local image features extracted from a lower layer in the CNN are used. Then an attention mechanism is proposed to choose the right image features to look at while generating different words in the caption. Similar attention mechanism is used in [75], but instead of using the attention model to pick local CNN features, it is used to

pick the right semantic concept, from the output of a semantic concept detector. Attention models extended to temporal domain have been applied to the video captioning task, in order to dynamically choose the right video feature [73]. Alternatively, a recurrent network is used to encode frame-level video features before inputting them to the language model in [64]. Then a standard LSTM language model is used to decode these features into a caption. 2.5 Datasets for Image and Video Captioning The rapid progress in automatic image and video captioning in the recent years has also been driven by the availability of large-scale datasets to train and test such models on. These captioning datasets have images or videos with one or more associated reference captions. The reference captions can be collected with large-scale human annotation using crowdsourcing tools such as Amazon Mechanical Turk or they can be mined from other related sources. One of the early datasets for image captioning was Pascal1K [47] consisting of 1000 images five human-annotated captions for each of them. Flickr8k [24] and Flickr30k [76] are relatively much larger datasets, consisting of 8,000 and 30,000 images, respectively. They also have five human-written captions for each image. Currently, the most popular and largest dataset for image captioning is the Microsoft Common Objects in Context (COCO) collection [39] with over 200,000 images and at least five human-written captions per image. There exists also an associated MS-COCO evaluation server, where researchers can upload their captions on the blind test dataset and compare the performance of their system to the state-of-the-art methods on a public leaderboard. Due to its size and availability of a standardized benchmark, all the image captioning experiments conducted on the MS-COCO dataset. Video captioning datasets are more difficult to collect and we can consequently see both automatic caption mining techniques and manual annotation used to collect them. YouTube corpus [8] consists of 2000 video clips with at least 27 textual descriptions for each video. The M-VAD [60] and MPII-MD

[50] datasets used in the first Large Scale Movie Description Challenge (LSMDC), were collected by extracting movie clips and transcribing the associated audio descriptions available in the movie DVDs. In total the LSMDC dataset contains about 100k clips from 202 movies and one reference caption associated with each clip. Creators of this dataset also provide an evaluation server to standardize the testing and comparison of performance on it. More recently released Microsoft Video to Text dataset (MSRVTT) [71] contains 10,000 short videos with 20 human annotated captions for each of them. This makes it the largest video captioning dataset in terms of video–caption pairs. The MSR-VTT dataset was also used in the recently concluded MSR-VTT video captioning challenge. We conduct our video captioning experiments on both the LSMDC and MSR-VTT datasets and also report results from our participation of the video captioning challenges associated with these datasets.

Chapter 3

Caption Generation Pipeline

In this chapter, we will examine in detail all the constituent parts of a baseline visual caption generation system. We adapt the model proposed in [65], the basis of the submission which jointly won the 1st Microsoft COCO captioning challenge in 2015, as our baseline model. Although the original model was proposed for generating captions for still images, the same architecture can be used for video captioning, by replacing the image feature extraction module with a video feature extraction module. Thus the discussion presented here is kept generic and specific details of features used for image or video captioning are discussed in Chapter 4. Then a discussion on the automatic evaluation metrics which are generally used to quantitatively rate the captions generated by the models is presented. The model presented in this chapter acts as the baseline against which we will compare the performance of the architectures and extensions proposed in the rest of the thesis.

3.1 Baseline Architecture

The baseline caption generation model consists of two stages: the visual feature extraction stage followed by a language model. The first stage consists of various techniques to extract descriptors of the visual contents of the input image or video. These descriptors are then represented as one or more vectors of fixed dimension. The language model then uses these feature vectors and generates a suitable caption to describe the image. This pipeline is illustrated

in Figure 3.1. In the following subsections, an overview of the different image features and the language model used in the baseline architecture is presented.

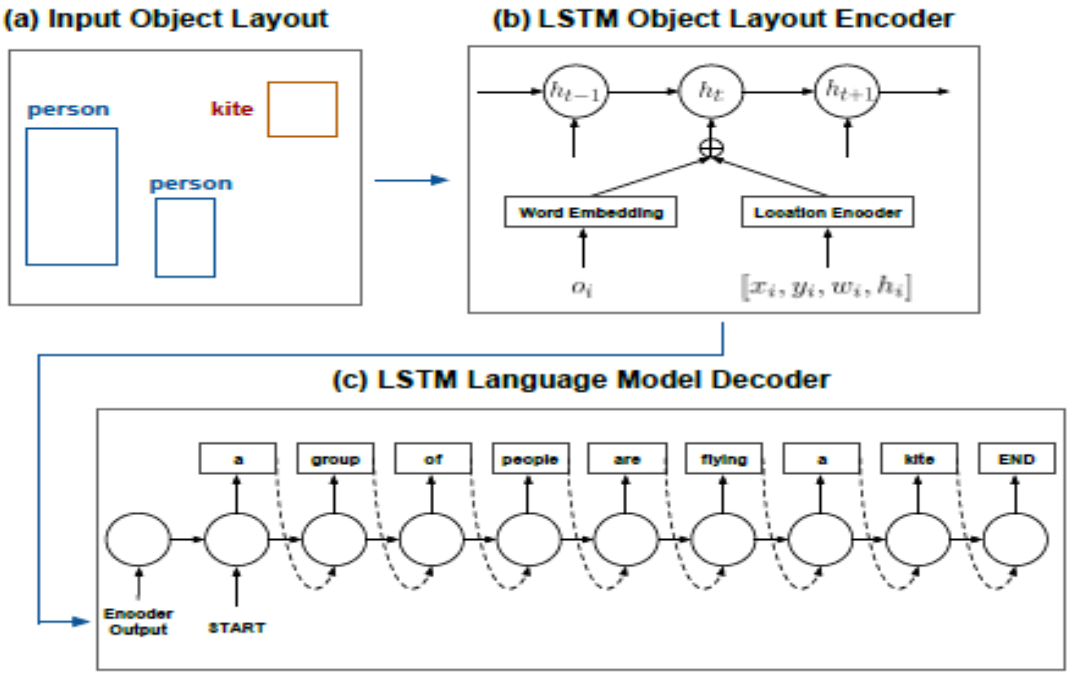


Figure 3.1 : Overview of our model. (a) object layout extracted from CNN. It consists of object categories and their corresponding bounding boxes. (b) Encoder : uses a two-stream RNN to encode the input object layout (c) Decoder: Uses a standard LSTM recurrent neural network to generate text.

3.1.1 Visual Feature Extraction

Baseline image feature. As discussed in Chapter 2, image features extracted from CNNs pre-trained on ImageNet have become ubiquitous in most image understanding tasks. Therefore, in

our baseline captioning model we use features extracted from GoogLeNet [59] as the image feature vector. More details of the feature extraction process are discussed in Chapter 4, but it suffices here to say that the feature vectors are formed by the activations of the 5th Inception module in GoogLeNet. Baseline video feature. As a very simple baseline feature vector for videos, we use the same GoogLeNet features as above, but extracted only from a single key frame of the video. We choose the key frame as the frame at the center of the video's time-span. The idea behind using this simple feature vector is to enable the video captioning baseline model to use the same pipeline as for images and to obtain a reasonable baseline against which more sophisticated feature extraction methods can be compared.

3.1.2 Language Model.

The next stage in the pipeline is a conditional language model which takes as input the visual features and generates a caption. The Long-Short Term Memory (LSTM) network [23] architecture has been a popular choice in the literature to model the probability of a sentence S, given an visual feature V, as P(S|V). The following two subsections contain a discussion of the LSTM cell in detail and on the way it is used to build a conditional language model.

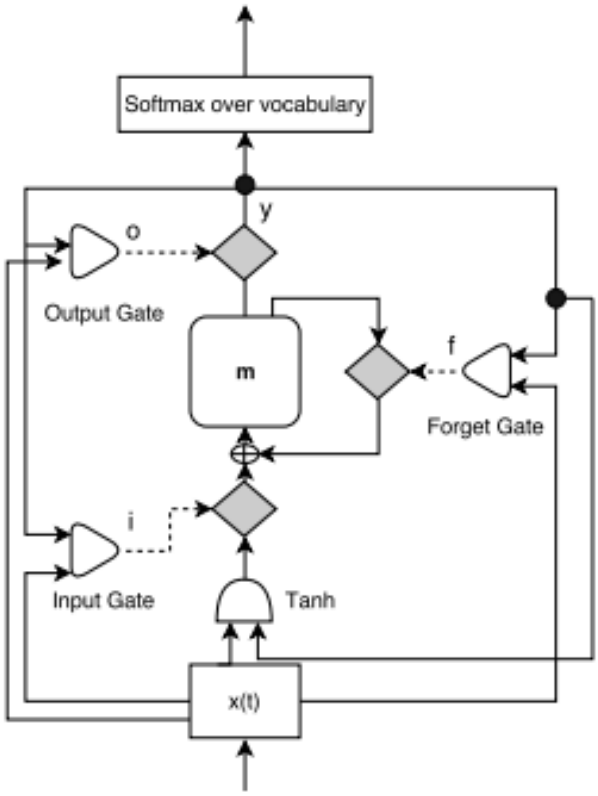


Figure 3.2: A single LSTM cell. Dotted lines to the rhombi indicate multiplications as gate controls and the solid lines depict the data flow. The triangles are sigmoid non-linearities.

Chapter 4 Visual Features

Finding good feature vector representations for the input images and videos is a very important task for successful design of a captioning system. Such a feature representation should be compact, but also able to encode all the information relevant for the task. For the image captioning task, the feature vector should capture all the objects in the image, their most essential properties such as color, their absolute position and relative location to each other, along with the type of the scene these objects are located in. In case of video captioning, apart from all the above mentioned information, the feature vector should also encode sufficient temporal information to enable recognizing actions, order of events, etc. In this chapter, we will study different visual features of images in order to improve the performance of the captioning system over the baseline presented in Chapter 3.

4.1 Image Feature Extraction

Activation values extracted from the deep Convolutional Neural Network (CNN) layers are the primary features used to represent images in the captioning models presented in this thesis. The CNN features are able to encode a rich variety of information, including scene context, object type, etc., as seen from its performance in the baseline model. However, this representation is still very dense and probably (as seen later in experiments) inefficient for the language model to be able to extract the information it needs to generate correct captions. It is also unclear to what extent these features encode multiple objects and object locations, since they

are trained on the ImageNet task involving recognizing a single object class. Thus, in a bid to improve the performance over the baseline model, language model is provided with additional features which explicitly encode presence of objects, scene types and object location. To achieve this, explicit object detectors and scene detectors are trained based on the CNN features. Additionally, features encoding object localization are constructed based on outputs from Faster Region-based Convolutional Neural Network (R-CNN) [48]. In the following subsections we will discuss the exact details of the processes used to extract all of the above features from input images.

4.1.1 Convolutional Neural Networks

Convolutional Neural Networks have in the recent years become the most widely used models for practically all tasks related to image classification and understanding. It is shown in [15] and [54] that activations of the fully connected layers of a CNN trained for image classification task act as a general feature representation of the image and can be successfully used to solve other tasks as well. In line with this, image features are extracted here from different CNN architectures pre-trained on two large datasets namely, ImageNet [12] and MIT Places [79], originally aimed for object and scene classification, respectively. The CNNs used here are based on the widely used GoogLeNet [59] and VGG [58] architectures. Both of these architectures achieved good results in the ILSVRC 2014 object classification challenge, finishing first and second, respectively. In this work, the GoogLeNet features are used both as a direct input to the language model and to train the place and object detector modules, while the VGG net features are only used for the latter.

Convolutional networks (LeCun, 1989), also known as convolutional neural networks , or CNNs, are a specialized kind of neural network for processing data that has a known grid-like topology. Examples include time-series data, which can be thought of as a 1-D grid taking samples at regular time intervals, and image data, which can be thought of as a 2-D grid of pixels. Convolutional networks have been tremendously successful in practical applications. The name "convolutional neural network" indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers. In this chapter, we first describe what convolution is. Next, we explain the motivation behind using convolution in a neural network. We then describe an operation called pooling, which almost all convolutional networks employ. Usually, the operation used in a convolutional neural network does not correspond precisely to the definition of convolution as used in other fields, such as engineering or pure mathematics. We describe several variants on the convolution function that are widely used in practice for neural networks. We also show how convolution may be applied to many kinds of data, with di erent numbers of dimensions. We then discuss means of making convolution more e cient. Convolutional networks stand out as an example of neuroscientific principles influencing deep learning.

We discuss these neuroscientific principles, then conclude with comments about the role convolutional networks have played in the history of deep learning. One topic this chapter does not address is how to choose the architecture of your convolutional network. The goal of this chapter is to describe the kinds of tools that convolutional networks provides. Research into convolutional network architectures proceeds so rapidly that a new best architecture for a given Benchmark is announced every few weeks to months, rendering it impractical to describe in print the best architecture. Nonetheless, the best architectures have consistently been composed of the building blocks described here.

4.1.1 Motivation

Convolution leverages three important ideas that can help improve a machine

Learning system : sparse interactions, parameter sharing and equivariant representations. Moreover, convolution provides a means for working with inputs of variable size. We now describe each of these ideas in turn. Traditional neural network layers use matrix multiplication by a matrix of parameters with a separate parameter describing the interaction between each input unit and each output unit. This means that every output unit interacts with every input unit. Convolutional networks, however, typically have sparse interactions (also referred to as sparse connectivity or sparse weights). This is accomplished by making the kernel smaller than the input. For example, when processing an image, the input image might have thousands or millions of pixels, but we can detect small, meaningful features such as edges with kernels that occupy only tens or hundreds of pixels. This means that we need to store fewer parameters, which both reduces the memory requirements of the model and improves its statistical e□ciency. It also means that computing the output requires fewer operations. These improvements in e□ciency are usually quite large.

If there are m inputs and n outputs, then matrix multiplication requires m ×n parameters, and the algorithms used in practice have O(m × n) runtime (per example). If we limit the number of connections each output may have to k, then the sparsely connected approach requires only k × n parameters and O(k × n) runtime. For many practical applications, it is possible to obtain good performance on the machine learning task while keeping k several orders of magnitude smaller than m. For graphical demonstrations of sparse connectivity, see figure 4.1 and figure 4.2 In a deep convolutional network, units in the deeper layers may indirectly interact with a larger portion of the input, as shown in figure 4.3. This allows the network to e□ciently describe complicated interactions between many variables by constructing such interactions from simple building blocks that each describe only sparse interactions

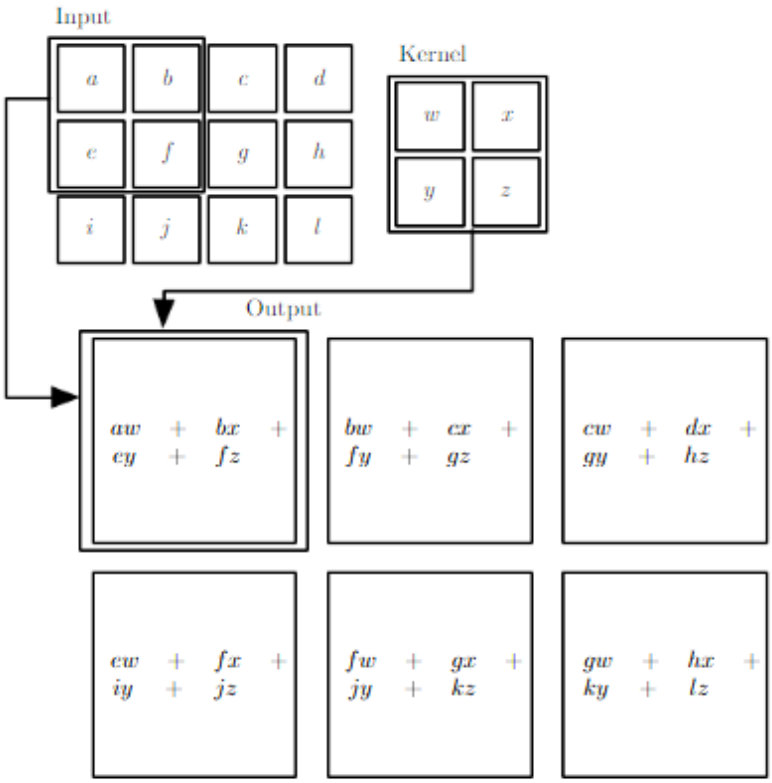


Figure 4.1 An example of 2-D convolution without kernel flipping. We restrict the output to only positions where the kernel lies entirely within the image, called "valid" convolution in some contexts. We draw boxes with arrows to indicate how the upper-left element of the output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor.

4.1.2 Parameter sharing: refers to using the same parameter for more than one function in a model

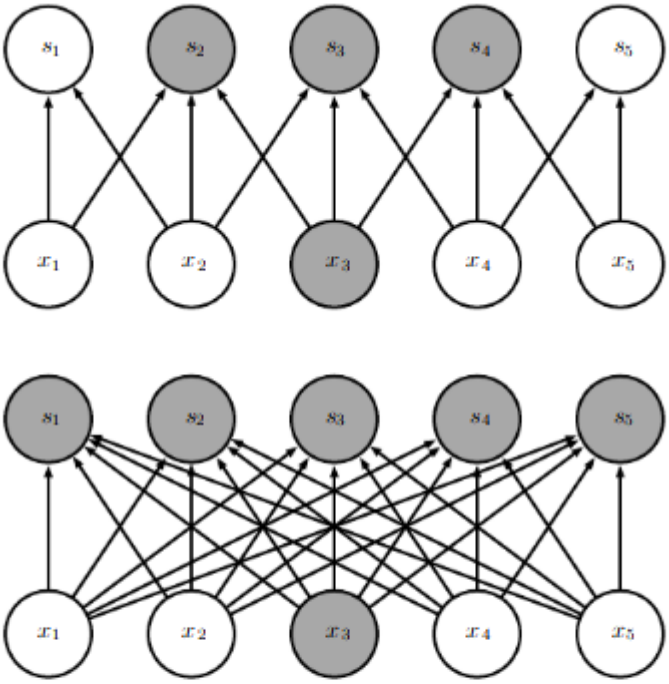


Figure 4.2 Sparse connectivity, viewed from below. We highlight one input unit, x_3, and highlight the output units in $\boldsymbol{s}$ that are affected by this unit. *(Top)*When $\boldsymbol{s}$ is formed by convolution with a kernel of width 3, only three outputs are affected by $\boldsymbol{x}$. *(Bottom)*When $\boldsymbol{s}$ is formed by matrix multiplication, connectivity is no longer sparse, so all the outputs are affected by x_3.

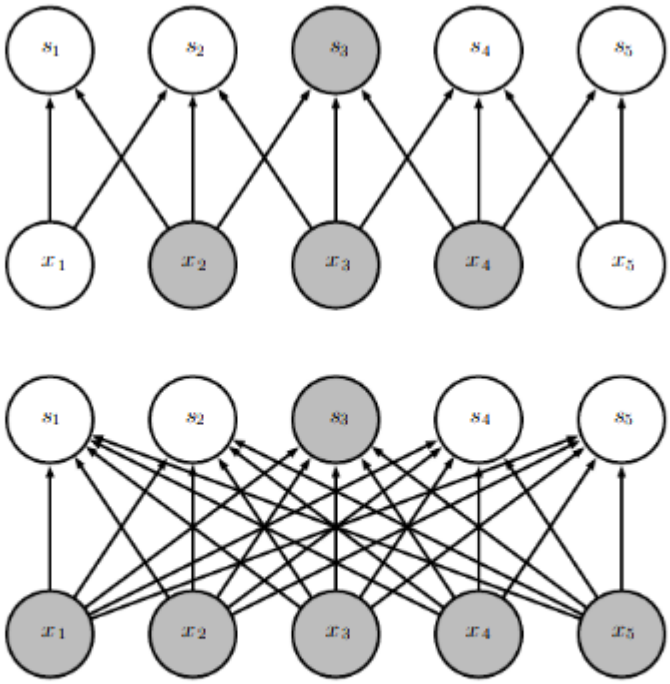


Figure 4.3 Sparse connectivity, viewed from above. We highlight one output unit, s_3, and highlight the input units in $\boldsymbol{x}$ that affect this unit. These units are known as the **receptive field** of s_3. *(Top)*When $\boldsymbol{s}$ is formed by convolution with a kernel of width 3, only three inputs affect s_3. *(Bottom)*When $\boldsymbol{s}$ is formed by matrix multiplication, connectivity is no longer sparse, so all the inputs affect s_3.

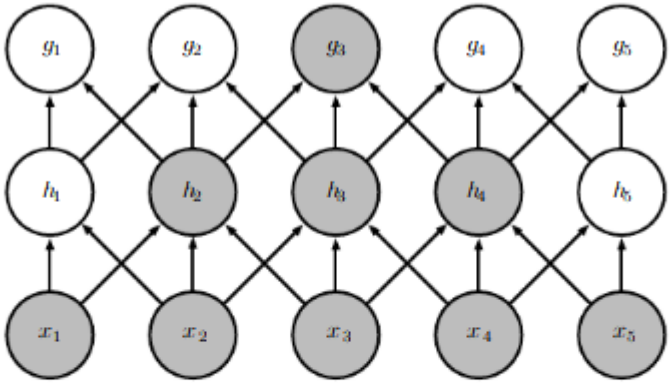


Figure 4.4 The receptive field of the units in the deeper layers of a convolutional network is larger than the receptive field of the units in the shallow layers. This effect increases if the network includes architectural features like strided convolution or pooling This means that even though *direct* connections in a convolutional net are very sparse, units in the deeper layers can be *indirectly* connected to all or most of the input image.

In a traditional neural net, each element of the weight matrix is used exactly once when computing the output of a layer. It is multiplied by one element of the input and then never revisited. As a synonym for parameter sharing, one can say that a network has tied weights, because the value of the weight applied to one input is tied to the value of a weight applied elsewhere. In a convolutional neural net, each member of the kernel is used at every position of the input (except perhaps some of the boundary pixels, depending on the design decisions regarding the boundary). The parameter sharing used by the convolution operation means that rather than learning a separate set of parameters for every location, we learn only one set. This does not a□ect the runtime of forward propagation—it is still O(k × n)—but it does further reduce the storage requirements of the model to k parameters. Recall that k is usually several orders of magnitude smaller than m. Since m and n are usually roughly the same size, k is practically insignificant compared to m ×n. Convolution is thus dramatically more e□cient than dense matrix multiplication in terms of the memory requirements and statistical e□ciency. For a graphical depiction of how parameter sharing works, see figure 4.5.

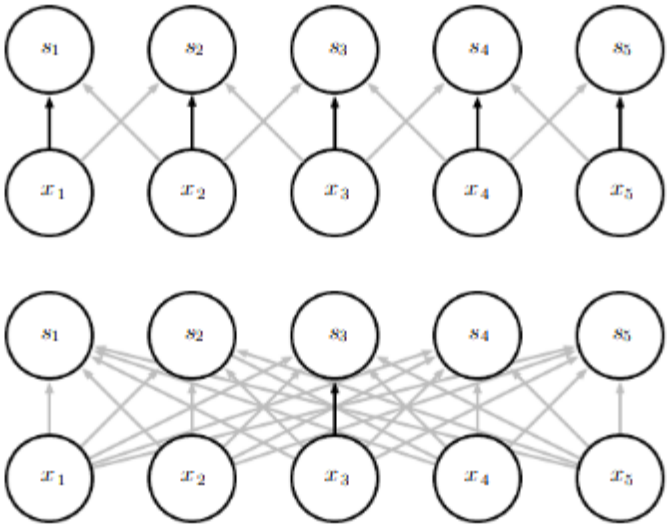


Figure 4.5 Parameter sharing. Black arrows indicate the connections that use a particular parameter in two different models. *(Top)*The black arrows indicate uses of the central element of a 3-element kernel in a convolutional model. Because of parameter sharing, this single parameter is used at all input locations. *(Bottom)*The single black arrow indicates the use of the central element of the weight matrix in a fully connected model. This model has no parameter sharing, so the parameter is used only once.

As an example of both of these first two principles in action, figure 4.6 shows how sparse connectivity and parameter sharing can dramatically improve the e□ciency of a linear function for detecting edges in an image. In the case of convolution, the particular form of parameter sharing causes the layer to have a property called equivariance to translation. To say a function is equivariant means that if the input changes, the output changes in the same way. Specifically, a function f(x) is equivariant to a function g if f(g(x)) =g(f(x)). In the case of convolution, if we let g be any function that translates the input, that is, shifts it, then the convolution function is equivariant to g. For example, let I be a function giving image brightness at integer coordinates. Let g be a function mapping one image function to another image function, such that I'=g(I) is the image function with I'(x, y) =I(x −1, y). This shifts every pixel of I one unit to the right. If we apply this transformation to I , then apply convolution, the result will be the same as if we

applied convolution to I', then applied the transformation to the output. When processing time-series data, this means that convolution produces a sort of timeline that shows when di□erent features appear in the input. If we move an event later in time in the input, the exact same representation of it will appear in the output, just later. Similarly with images, convolution creates a 2-D map of where certain features appear in the input. If we move the object in the input, its representation will move the same amount in the output. This is useful for when we know that some function of a small number of neighboring pixels is useful when applied to multiple input locations. For example, when processing images, it is useful to detect edges in the first layer of a convolutional network. The same edges appear more or less everywhere in the image, so it is practical to share parameters across the entire image. In some cases, we may not wish to share parameters across the entire image. For example, if we are processing images that are cropped to be centered on an individual's face, we probably want to extract di□erent features at di□erent locations—the part of the network processing the top of the face needs to look for eyebrows, while the part of the network processing the bottom of the face needs to look for a chin. Convolution is not naturally equivariant to some other transformations, such as changes in the scale or rotation of an image. Other mechanisms are necessary for handling these kinds of transformations. Finally, some kinds of data cannot be processed by neural networks defined by matrix multiplication with a fixed-shape matrix. Convolution enables processing of some of these kinds of data.

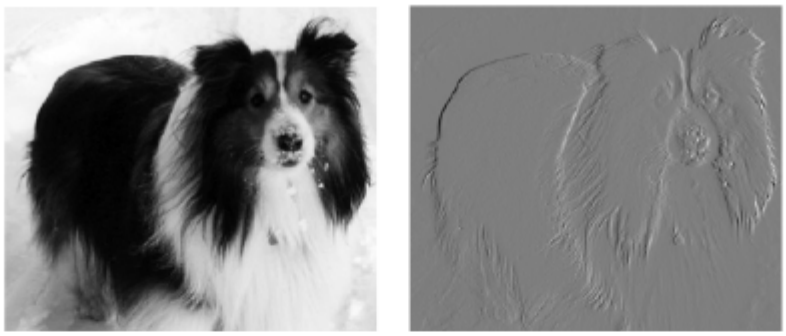


Figure 4.6 Efficiency of edge detection. The image on the right was formed by taking each pixel in the original image and subtracting the value of its neighboring pixel on the left. This shows the strength of all the vertically oriented edges in the input image, which can be a useful operation for object detection. Both images are 280 pixels tall. The input image is 320 pixels wide, while the output image is 319 pixels wide. This transformation can be described by a convolution kernel containing two elements, and requires $319 \times 280 \times 3 = 267,960$ floating-point operations (two multiplications and one addition per output pixel) to compute using convolution. To describe the same transformation with a matrix multiplication would take $320 \times 280 \times 319 \times 280$, or over eight billion, entries in the matrix, making convolution four billion times more efficient for representing this transformation. The straightforward matrix multiplication algorithm performs over sixteen billion floating point operations, making convolution roughly 60,000 times more efficient computationally. Of course, most of the entries of the matrix would be zero. If we stored only the nonzero entries of the matrix, then both matrix multiplication and convolution would require the same number of floating-point operations to compute. The matrix would still need to contain $2 \times 319 \times 280 = 178,640$ entries. Convolution is an extremely efficient way of describing transformations that apply the same linear transformation of a small local region across the entire input. Photo credit: Paula Goodfellow.

4.1.2.1 GoogLeNet

The main idea behind the GoogLeNet [59] architecture is to use small dense structures like 1×1, 3×3 convolutions to mimic a large sparse layer. For this purpose they utilize the Inception modules consisting of 1×1, 3×3 and 5×5 convolutions and maximum pooling layers. This network achieved the top-5 error rate of 6.67% in the 1000 class ILSVRC 2014 Classification Challenge, finishing first in the competition. We use two different versions of the GoogLeNet features in our experiments. Features from GoogLeNet trained on ImageNet [12]

dataset are used as direct input to our language model (referred to as ”gCNN ” in the rest of the text), whereas features from the GoogLeNet trained on MIT Places [79] data (referred to as ”pCNN ” in the rest of the text) are used to construct scene recognition features which are used as auxiliary inputs to our language model. To extract gCNN features, we use the activations from the 5th Inception module, having the dimensionality of 1024. We augment these features with the reverse spatial pyramid pooling proposed in [19] with two scale levels. The first scale is just the full image rescaled to the size of 224 × 224. The second level consists of a 3 × 3 grid of overlapping patches of size 128 × 128 with stride of 64, and horizontal flipping. The activations of these regions are then reduced to a single 1024 dimensional feature vector by using average pooling or by just using the central crop. Finally, the activations of the two scales are concatenated resulting in 2048-dimensional features. Note that due to two different pooling methods on the second scale, we obtain two somewhat different feature vectors of 2048 dimensions from the same network. Our final gCNN feature vector of size 4096 dimensions is obtained by concatenating these two feature vectors. This is also the feature vector used as the input to our baseline image captioning model. The same procedure described above for the ImageNet trained GoogLeNet has also been followed with the Places data trained GoogLeNet, with the exception that instead of the Inception module, the 3rd classification branch has been used as the activation layer where the feature vectors have been extracted. In this case, in addition to the mean and center pooling in the second scale of reverse spatial pyramid, we also use maximum pooling, and thus obtain three different features with the dimensionality of 2048 each. Note that the term pCNN refers collectively to this set of three features.

4.1.2.2 VGG Network

VGG network was introduced in [58], where the authors study the effect of depth on the performance of convolutional networks. The salient feature of this architecture is the exclusive use of small 3 × 3 and 2 × 2 convolutional filters throughout the network. This helps in keeping the number of parameters small even with the increased depth. This network achieves the top-5 error rate of 7.3% in the 1000 class ILSVRC 2014 Classification Challenge, finishing second in the competition. Two variants of the VGG net, namely the 16- and 19-layered ones from [58], are used in the experiments reported here. From both the variants, we extract the activations of the network on the second fully-connected 4096- dimensional fc7 layer for the given input images whose aspect ratio is distorted to a square. Ten regions, as suggested in [34], are extracted from all images and average pooling of the region-wise features are used to generate the final features.

4.1.2 Object Detectors

As mentioned before, the CNN features are augmented with explicit object detector features where each dimension represents the presence or absence of one of the 80 object categories defined in the COCO dataset [39]. These 80 categories consists of common object types such as person, car, bus, dog, cat, table, chair, pizza, banana, laptop, etc. To construct the explicit object detector features, 80 separate SVM classifiers [10] are trained on the COCO 2014 [39] training set to detect each of these 80 object categories. Image features extracted using the previously described five CNN-based ImageNet-trained VGG and GoogLeNet features are used as input to the 80 object detector SVMs. In particular, we here utilized linear SVMs with homogeneous kernel maps [62] of second order to approximate the intersection kernel.

Furthermore, we used two rounds of hard negative mining [37] and sampled 5 000 negative examples on each round. For each image we thus have 15 SVM outputs for each class (five features times initial and two hard negative trained models) that we combine with simple arithmetic mean in the late fusion stage. The 80 fused output values, one for each object category, are then concatenated to form a class membership vector for each image. These vectors we optionally use as inputs to the LSTM network and we denote it as “SVM80 ” in the rest of the thesis.

4.1.3 Scene Detectors

In order to provide the language model with explicit information on the visual environment or the scene type of the images, we used the SUN Scene Categorization Benchmark database [70] and [69] to create a bank of visual detectors specialized for scene recognition. The version of the database we used contains 108,756 images associated with one of 397 scene categories. The 397 categories include three major classes — namely indoor, outdoor-natural and outdoor-man-made — and common scene types including kitchen, livingroom, shower, tennis-court, courtroom, beach, dock, airfield, dam etc. We extracted both ImageNet data trained and MIT Places data trained GoogLeNet CNN features, as described in the previous section, for the images in the SUN database. We used features of all the images (not only the training split) for training Radial Basis Function (RBF) Support Vector Machines (SVMs) with the LIBSVM software [7]. As we had three slightly different versions of each of the feature types, we obtained the total of six SVM detectors for each scene category. We applied each of the detectors to the images of the COCO dataset and used the simple arithmetic mean for the late fusion of the

detector outputs. The concatenation of the fused category-wise detector outputs results in 397-dimensional feature vectors for the respective images. These feature vectors are referred to as "SUN397 " in the rest of the thesis. Figure 4.1 shows the top five detected scene categories using the SUN397 features for an image from the COCO validation set.

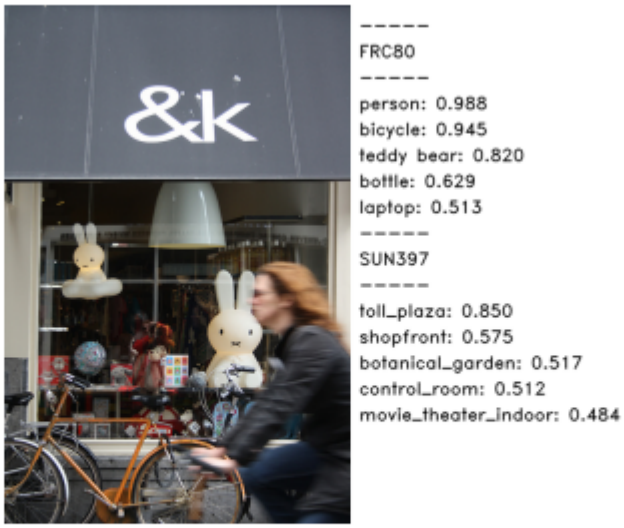


Figure 4.7 *Visualization of top five detected objects (FRC80) and scene types (SUN397) along with the associated confidence scores for an image from the MS-COCO validation set.*

Chapter 5 Language Model (Sequence Modeling): RNN

In this chapter, we will look at the core concepts underlying recurrent neural networks, the problems they face(namely the problem of Vanishing gradient and Exploding gradient), and the solution to fight the problems: Gradient Recurrent Unit(GRU) and Long Short term Memory(LSTM). We will have the chance to go through the inner workings of each cells in recurrent network. We will also see how gradient descent is performed to train these networks. We will develop these theories to understand Encoder-Decoder sequence to sequence architecture, which is the topic of this thesis.

Recurrent Neural Networks(RNNs) are the family of neural networks for handling sequential data, like text data. As we saw in chapter 4 that Convolutional Neural Networks are good at handling grid of values, in the same way RNNs specialise in dealing with sequential data.

Why RNN? To understand why we need a recurrent network instead of conventional fully connected neural network, we will have to go back to the statistical modeling ideas of '80s : *The idea of parameter sharing across different parts of a model.* Parameter sharing makes it possible to apply RNN models to sequences of varying lengths. Parameter sharing also facilitates generalisation on new unseen data set. For example, consider that a machine learning system is given sentences, like (a) I reached airport at 9.12 am (b) At 9.12 am, I reached airport, and asked

to extract the time at which person reached airport, we would want our system to return ‘9.12 am’ for both the sentences , irrespective of whether the word appears first in the sentence or at last. A traditional fully connected neural network would have learnt different rules for different positions because it doesn’t share parameters.

For simplicity, we refer to RNNs as operating on a sequence that contains vectors x(t) with the time step index t □ (1 , τ) . In practice, recurrent networks usually operate on mini batches of such sequences, with a di□erent sequence length τ for each member of the minibatch. We have omitted the minibatch indices to simplify notation. Moreover, the time step index need not literally refer to the passage of time in the real world. Sometimes it refers only to the position in the sequence.

In the following subsection we will see ideas of unfolding a recursive computation into a computational graph. But, let’s first understand what a computational graph is ..

5.1 Computational Graph

Using a computational graph one can formalize steps of calculations, such as mapping from input to output or calculation involving losses. Few examples of computational graph can be seen in figure 5.1.

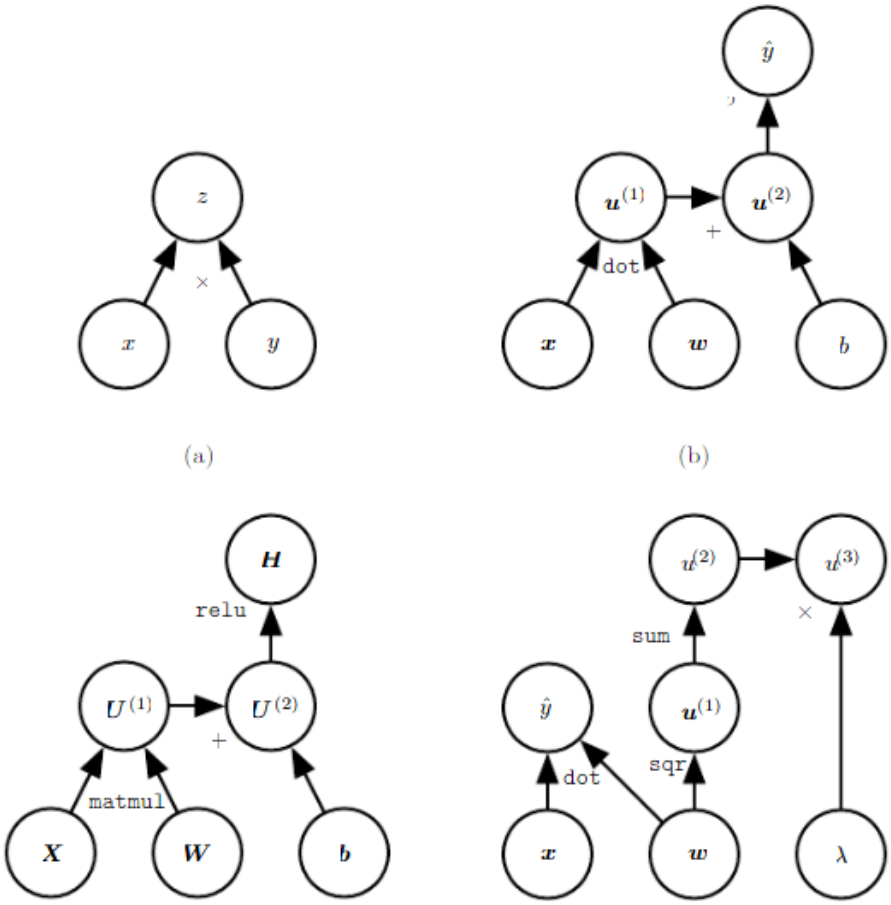


Figure 5.1: Examples of computational graphs.

5.2 Unfolding a computational graph

Consider a recurrent equation $\mathbf{s^{(t)} = f(\ s^{(t-1)}; \Theta)}$, this calculation can be unfolded, for example, for t =3:

$$s^{(3)} = f(\ s^{(2)}; \Theta)$$

$$= f(f(\ s^{(1)}; \Theta); \Theta)$$

Illustration can be seen in figure 5.2

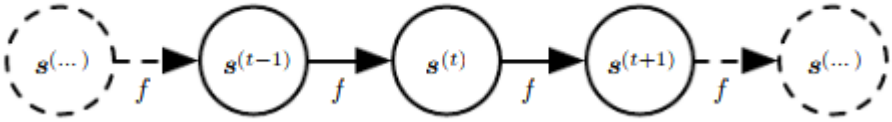


Figure 5.2: Unfolded computational graph. Node represents state at some time t. Function 'f' maps state at 't' to state at 't+1'.

Another Example, more relevant to our thesis, is :

$$\mathbf{h}^{(t)} = \mathbf{f}(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \Theta)$$

Where $\mathbf{x}^{(t)}$ is input sequence and $\mathbf{h}^{(t)}$ is hidden state. Figure 5.3

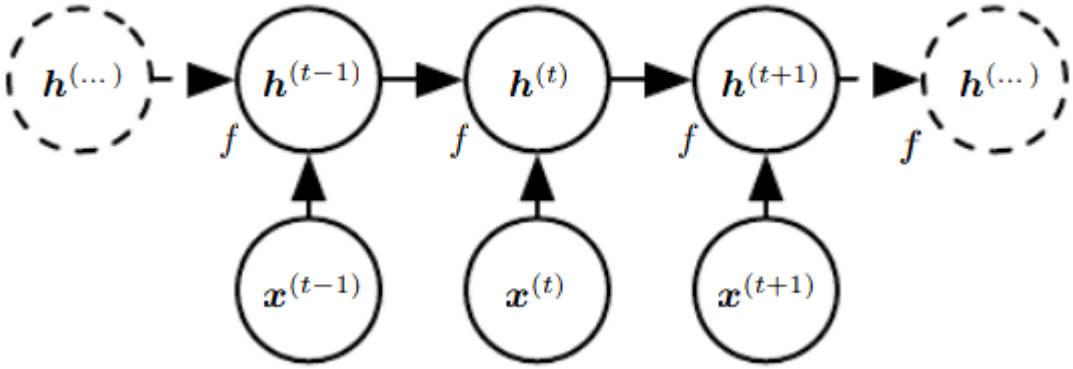


Figure 5.3 Unfolded computational graph, each node is associated with each particular time instant

When an RNN is trained to predict a future value depending on past values, the network learns to use $h^{(t)}$ as a kind of lossy summary for past sequence (upto 't') . The summary is necessarily lossy because $\mathbf{h}^{(t)}$ is a fixed length vector, whereas past sequence could be of any arbitrary length.

Other than explicitly describing how to perform computation, unrolled graph has following advantages:

1. The learned model always has the same input size, regardless of sequence length.
2. It allows to use same transition function '*f*' with same parameters at every time step.

5.3 Recurrent Neural Networks

Now that we understand unrolling of graph and parameter sharing, we can design wide variety of Recurrent neural network.

1. Recurrent networks that gives an output at each time step and they have recurrent connections between hidden units. (explained in 5.3.1).
2. Recurrent networks that give output at each time step and they have recurrent connections from output to next time step input.(explained in 5.3.2).
3. Recurrent network that reads an entire sequence first and then gives just a single output. They have recurrent connections between hidden units. (explained in 5.3.3)

5.3.1 Recurrent networks that gives an output at each time step and they have recurrent connections between hidden units.

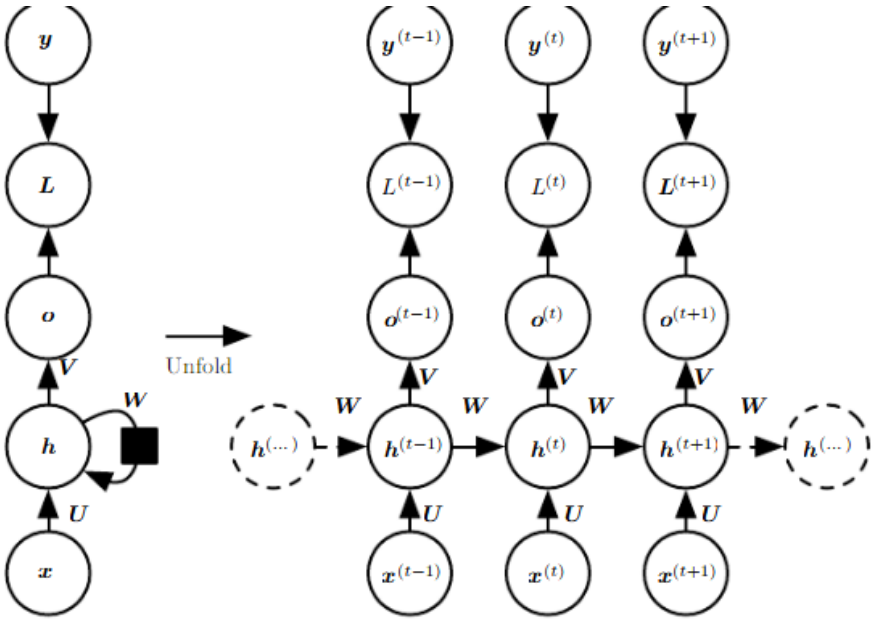


Figure 5.4 The computational graph to compute the training loss of a recurrent network that maps an input sequence of $\boldsymbol{x}$ values to a corresponding sequence of output $\boldsymbol{o}$ values. A loss L measures how far each $\boldsymbol{o}$ is from the corresponding training target $\boldsymbol{y}$. When using softmax outputs, we assume $\boldsymbol{o}$ is the unnormalized log probabilities. The loss L internally computes $\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{o})$ and compares this to the target $\boldsymbol{y}$. The RNN has input to hidden connections parametrized by a weight matrix $\boldsymbol{U}$, hidden-to-hidden recurrent connections parametrized by a weight matrix $\boldsymbol{W}$, and hidden-to-output connections parametrized by a weight matrix $\boldsymbol{V}$. Equation below defines forward propagation in this model. *(Left)* The RNN and its loss drawn with recurrent connections. *(Right)* The same seen as a time-unfolded computational graph, where each node is now associated with one particular time instance.

We now develop equations for forward propagation for the architecture shown above in figure 5.4. For each time step in t □ (1 , τ), we can think that the architecture is doing the following calculation:

$\mathbf{a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}}$

$\mathbf{h^{(t)} = tanh(a^{(t)})}$

$\mathbf{o^{(t)} = c + Vh^{(t)}}$

Where b,c are bias vectors and W,U,V are weight matrices.

$o^{(t)}$ can be thought of as a unnormalised log probabilities, which can then be passed through a softmax to obtain a vector $\hat{y}$ which is normalised probability over output.

$\mathbf{\hat{y}^{(t)} = softmax(o^{(t)})}$

Loss : The total loss is given by losses over all time steps:

$$
\begin{aligned}
& L\left(\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}\}, \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(\tau)}\}\right) \\
&= \sum_{t} L^{(t)} \\
&= -\sum_{t} \log p_{\text{model}}\left(y^{(t)} \mid \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}\}\right),
\end{aligned}
$$

Where entry for $y^{(t)}$ can be read from model's output vector $\hat{y}^{(t)}$.

In summary, The model takes input, makes forward propagation from left to right in an unrolled graph like shown in figure 5.4, followed by a back propagation pass from right to left. The backward pass applied to unrolled graph is called backpropagation through time(BPTT).

5.3.2 Recurrent networks that give output at each time step and they have recurrent connections from output to next time step input.

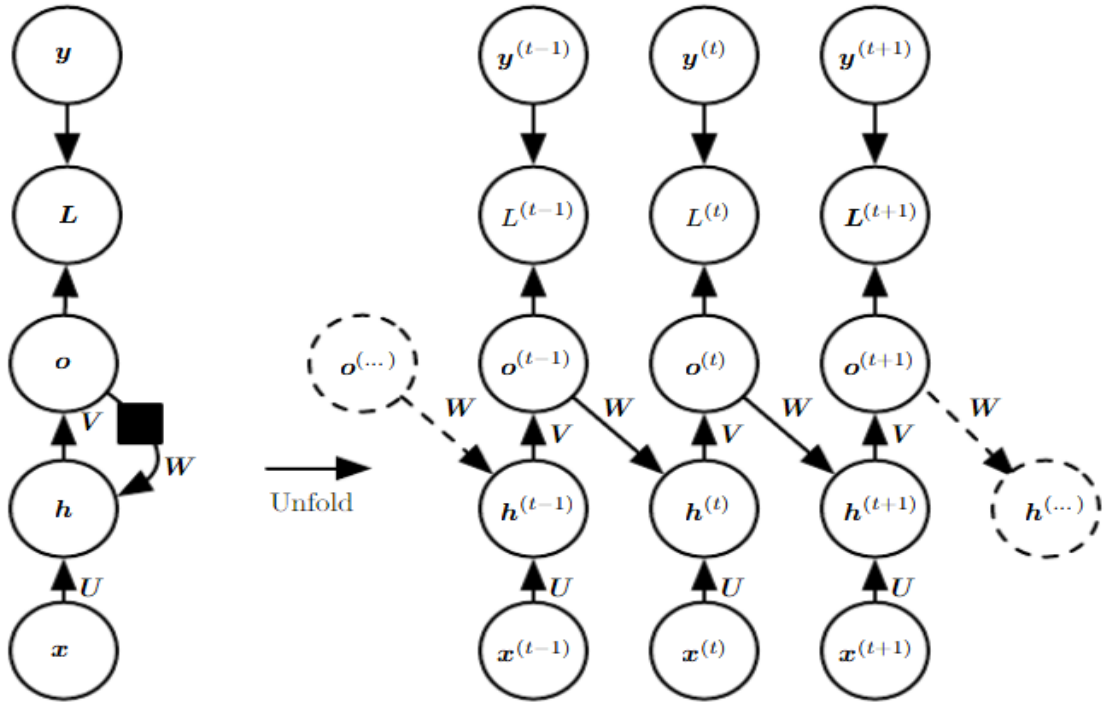


Figure 5.5 : An RNN whose only recurrence is the feedback connection from the output to the hidden layer. At each time step t, the input is $\boldsymbol{x}_t$, the hidden layer activations are $\boldsymbol{h}^{(t)}$, the outputs are $\boldsymbol{o}^{(t)}$, the targets are $\boldsymbol{y}^{(t)}$, and the loss is $L^{(t)}$. *(Left)* Circuit diagram. *(Right)* Unfolded computational graph. Such an RNN is less powerful (can express a smaller set of functions) than those in the family represented by figure 5.4 . The RNN in figure 5.4 can choose to put any information it wants about the past into its hidden representation $\boldsymbol{h}$ and transmit $\boldsymbol{h}$ to the future. The RNN in this figure is trained to put a specific output value into $\boldsymbol{o}$, and $\boldsymbol{o}$ is the only information it is allowed to send to the future. There are no direct connections from $\boldsymbol{h}$ going forward. The previous $\boldsymbol{h}$ is connected to the present only indirectly, via the predictions it was used to produce. Unless $\boldsymbol{o}$ is very high-dimensional and rich, it will usually lack important information from the past. This makes the RNN in this figure less powerful, but it may be easier to train because each time step can be trained in isolation from the others, allowing greater parallelization during training,

5.3.3 Recurrent network that reads an entire sequence first and then gives just a single output. They have recurrent connections between hidden units.

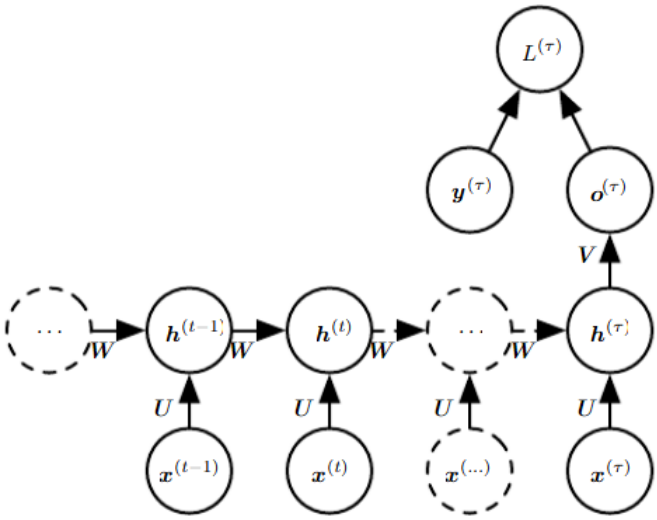


Figure 5.6 : Time-unfolded recurrent neural network with a single output at the end of the sequence. Such a network can be used to summarize a sequence and produce a fixed-size representation used as input for further processing. There might be a target right at the end (as depicted here), or the gradient on the output $o^{(t)}$ can be obtained by back-propagating from further downstream modules.

5.4 Computing Gradient in Recurrent Neural Network

Computation of gradient through RNN is simple, we can simply apply backpropagation algorithm to an unrolled graph as described in chapter 3. For illustration let's see how backpropagation is applied to unfolded graph such as to those described in section 5.3(figure 5.4).

For each node(N) in the graph, we need to calculate $\nabla_N L$ recursively , based on the gradients computed at nodes that follow it in the graph. Let's start the recursion with the nodes immediately preceding the final loss.

$$\frac{\partial L}{\partial L^{(t)}} = 1.$$

As we have seen in 5.3.1, $o^{(t)}$ can be thought of as an unnormalised log probabilities, which can then be passed through a softmax to obtain a vector $\hat{y}$ which is normalised probability over output. The gradient $\nabla_{o(t)} L$ on outputs at time step t , for all i,t is following

$$(\nabla_{\boldsymbol{o}^{(t)}} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - \mathbf{1}_{i,y^{(t)}}.$$

At final time step τ, $h^{(\tau)}$ only has $o^{(\tau)}$ as a descendent, so gradient is

$$\nabla_{\boldsymbol{h}^{(\tau)}} L = \boldsymbol{V}^\top \nabla_{\boldsymbol{o}^{(\tau)}} L.$$

Now we can iterate back through time from τ-1 to 1.

$$\begin{aligned}\nabla_{\boldsymbol{h}^{(t)}} L &= \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}}\right)^\top (\nabla_{\boldsymbol{h}^{(t+1)}} L) + \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}}\right)^\top (\nabla_{\boldsymbol{o}^{(t)}} L) \\ &= \boldsymbol{W}^\top (\nabla_{\boldsymbol{h}^{(t+1)}} L) \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t+1)}\right)^2\right) + \boldsymbol{V}^\top (\nabla_{\boldsymbol{o}^{(t)}} L),\end{aligned}$$

We have gradients of internal nodes,we can obtain gradients of parameter nodes. It is important to remember that parameters are shared across many time steps so we should be careful with our calculus operations.

$$
\begin{aligned}
\nabla_{\boldsymbol{c}} L &= \sum_t \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{c}}\right)^{\top} \nabla_{\boldsymbol{o}^{(t)}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L, \\
\nabla_{\boldsymbol{b}} L &= \sum_t \left(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}^{(t)}}\right)^{\top} \nabla_{\boldsymbol{h}^{(t)}} L = \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \nabla_{\boldsymbol{h}^{(t)}} L, \\
\nabla_{\boldsymbol{V}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}}\right) \nabla_{\boldsymbol{V}} o_i^{(t)} = \sum_t \left(\nabla_{\boldsymbol{o}^{(t)}} L\right) \boldsymbol{h}^{(t)\top}, \\
\nabla_{\boldsymbol{W}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{W}^{(t)}} h_i^{(t)} \\
&= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \left(\nabla_{\boldsymbol{h}^{(t)}} L\right) \boldsymbol{h}^{(t-1)\top}, \\
\nabla_{\boldsymbol{U}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{U}^{(t)}} h_i^{(t)} \\
&= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \left(\nabla_{\boldsymbol{h}^{(t)}} L\right) \boldsymbol{x}^{(t)\top},
\end{aligned}
$$

Obviously, we won't compute gradient with respect to input $x^{(t)}$ for training because it doesn't have any parameters.

All the recurrent neural networks that we have seen so far were causal networks that means the state at t captures information from past t-1 steps but not from future time steps. If somehow we could include information from future states, then the performance is bound to improve. In the following subsections we will see how we can improve performance by adding bidirectionality in recurrent neural network.

5.5 Bidirectional Recurrent Neural Network

The idea for bidirectional Recurrent Neural Networks is just an extension of Recurrent Neural Network that we have seen in the previous sections. In bidirectional, we combine two RNNs, one that propagates information forward through time, and one that propagates information backward through time. In Figure 5.7 the ‘h’ recurrence propagates information forward (Starting from beginning to end) in time while ‘g’ propagtes the information backwards(Starting from end to beginning).

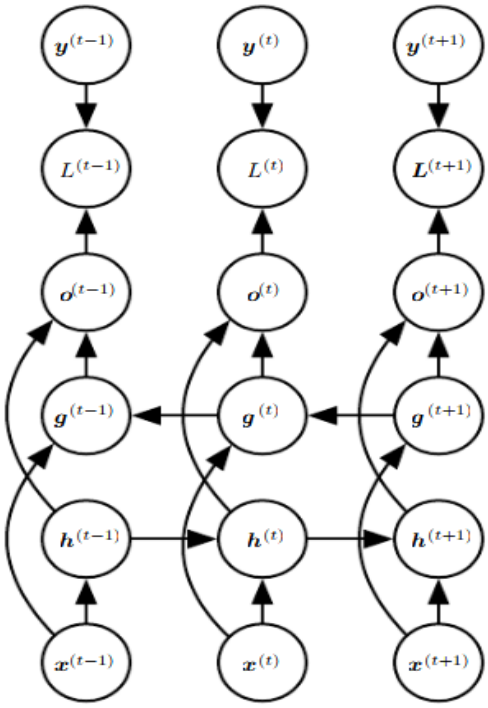


Figure 5.7 : Computation of a typical bidirectional recurrent neural network, meant to learn to map input sequences $\boldsymbol{x}$ to target sequences $\boldsymbol{y}$, with loss $L^{(t)}$ at each step t. The $\boldsymbol{h}$ recurrence propagates information forward in time (toward the right), while the $\boldsymbol{g}$ recurrence propagates information backward in time (toward the left). Thus at each point t, the output units $\boldsymbol{o}^{(t)}$ can benefit from a relevant summary of the past in its $\boldsymbol{h}^{(t)}$ input and from a relevant summary of the future in its $\boldsymbol{g}^{(t)}$ input.

5.6 Difficulty of Training over many time steps/ The Problem of Long-Term Dependencies

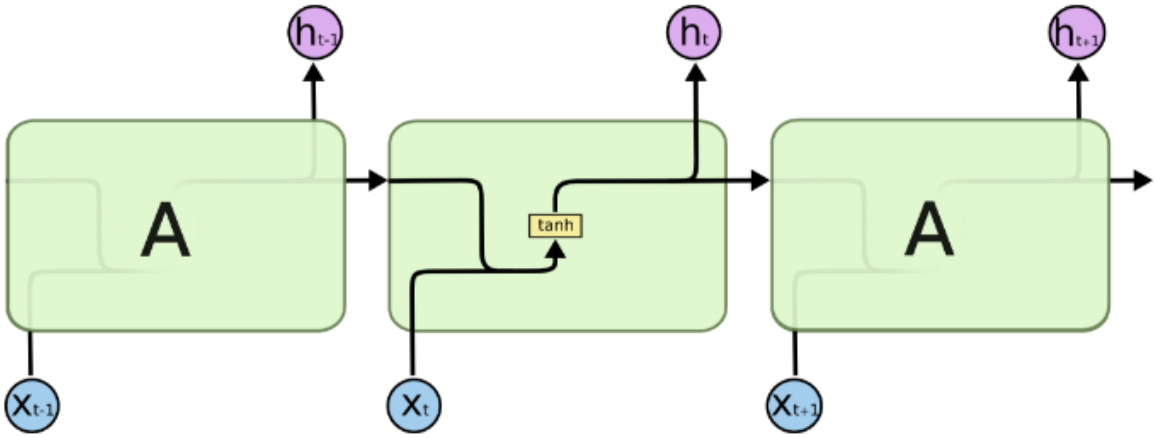
While training a long-running RNNs one of the problem that is faced is the fact that memory of first input gradually tends to fade by the time it reaches fourth or fifth memory cell. For example if we wish to use our model for sentiment analysis and a reviewer has written something like ‘Absolutely loved the movie, despite poor performances by side-actors’. If we use a simple RNN model to predict rating for the review, the model will not tend to do well because it would forget the beginning part (‘Absolutely loved the..’) and predict according to last few words.

To solve these kinds of problems, various types of cells with long distance memory were introduced. We will go through two of the most widely used cells: Long Short Term Memory networks – usually just called “LSTMs” and Gradient Recurrent Units aka ‘GRUs’.

5.6.1 Long Short Term Memory networks(LSTMs)

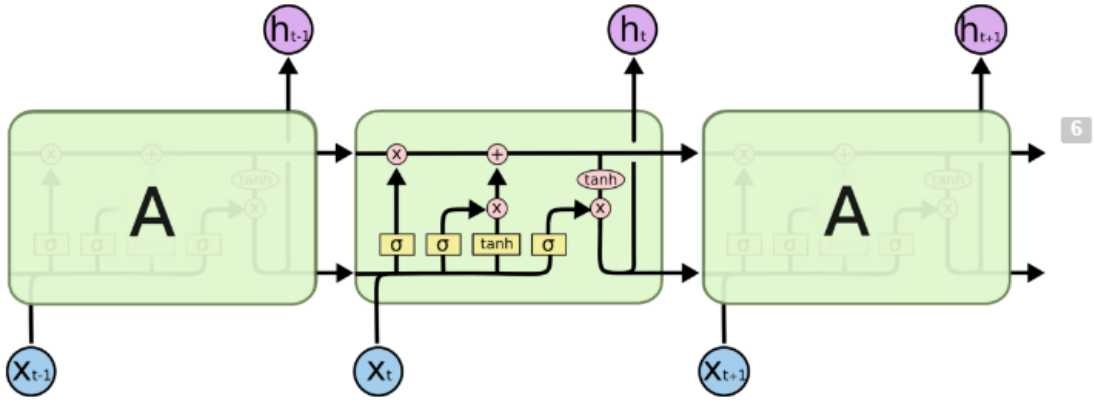
Long Short Term Memory networks – usually just called “LSTMs” are special kind of Recurrent Neural Network, capable of handling long term dependencies. It was introduced by Hochreiter and Schmidhuber in 1997, and further improved by many(see reference), and is now being widely used.

Let’s go through the explanation of LSTM. The following figure is a figure of simple RNN we see that repeating module has simple structure, such as only single tanh layer.



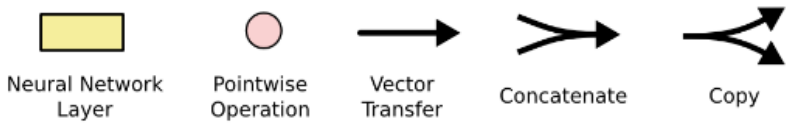
The repeating module in a standard RNN contains a single layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



The repeating module in an LSTM contains four interacting layers.

Let's get ourselves familiarized with the notations used:

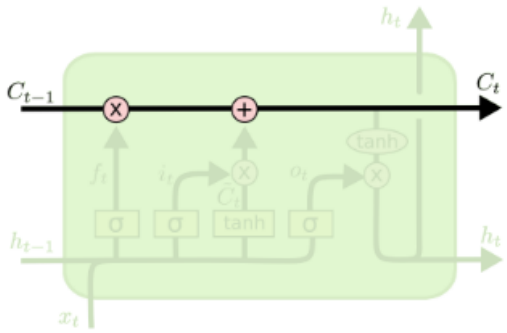


In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.

5.6.1.1 Core Ideas behind Long short term memories.

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.

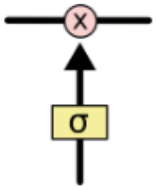
The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.



4

The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.



4

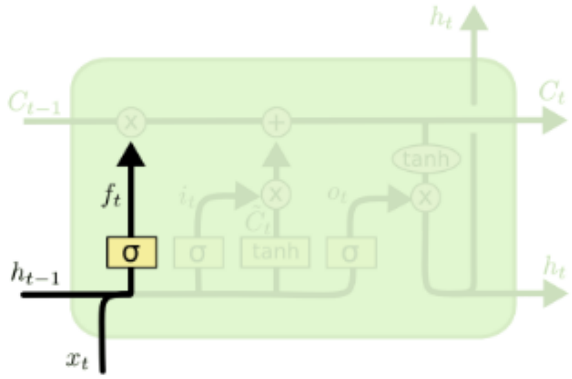
The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through!"

An LSTM has three of these gates, to protect and control the cell state.

5.6.1.2 LSTM Step-by-Step Walk Through

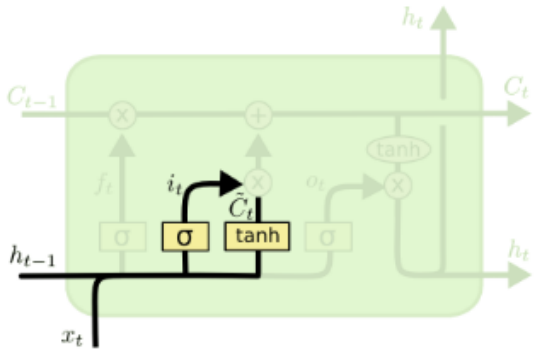
The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at $\boldsymbol{h_{t-1}}$ and $\mathbf{x_t}$, and outputs a number between 0 and 1 for each number in the cell state $\boldsymbol{C_{t-1}}$. A 1 represents "completely keep this" while a 0 represents "completely get rid of this."

Let's go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.



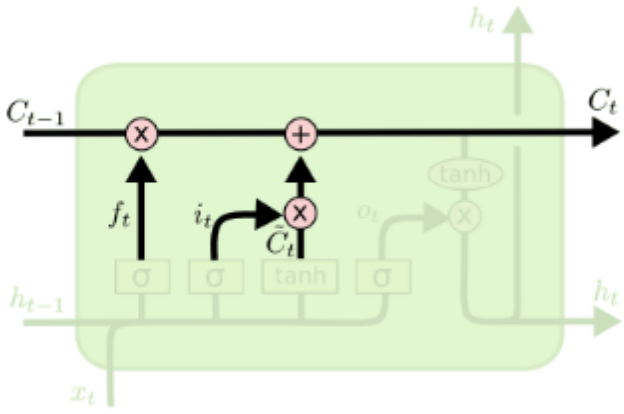
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, $\hat{C}_t$, that could be added to the state. In the next step, we'll combine these two to create an update to the state. In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

It's now time to update the old cell state, C_{t-1}, into the new cell state C_t. The previous steps already decided what to do, we just need to actually do it. We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add i_t * C^t. This is the new candidate values, scaled by how much we decided to update each state value. In the case of the language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in the previous steps.

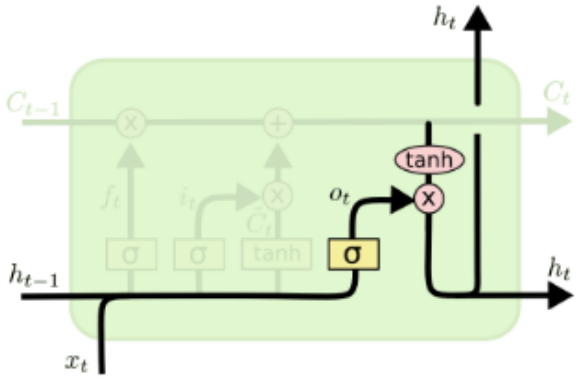


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the

cell state we're going to output. Then, we put the cell state through tanh(to push the values to be between −1 and −1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

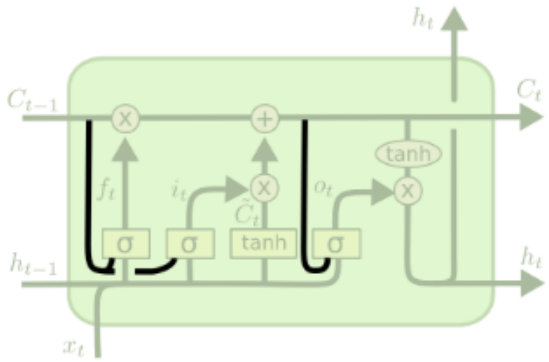
For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

5.6.1 .3 Variants on Long Short Term Memory

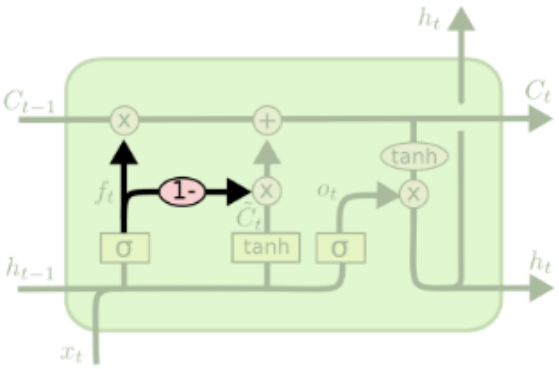
What I've described so far is a pretty normal LSTM. But not all LSTMs are the same as the above. In fact, it seems like almost every paper involving LSTMs uses a slightly different version. The differences are minor, but it's worth mentioning some of them. One popular LSTM variant, introduced by Gers & Schmidhuber (2000), is adding "peephole connections." This means that we let the gate layers look at the cell state.



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

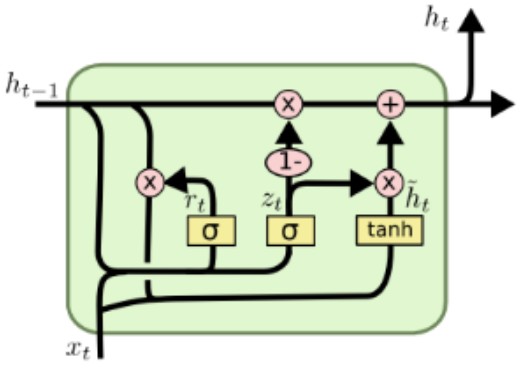
The above diagram adds peepholes to all the gates, but many papers will give some peepholes and not others.

Another variation is to use coupled forget and input gates. Instead of separately deciding what to forget and what we should add new information to, we make those decisions together. We only forget when we're going to input something in its place. We only input new values to the state when we forget something older.



$$C_t = f_t * C_{t-1} + (\mathbf{1} - \boldsymbol{f_t}) * \tilde{C}_t$$

A slightly more dramatic variation on the LSTM is the Gated Recurrent Unit, or GRU, introduced by Cho, et al. (2014). It combines the forget and input gates into a single "update gate." It also merges the cell state and hidden state, and makes some other changes. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

These are only a few of the most notable LSTM variants. There are lots of others, like Depth Gated RNNs by Yao, et al. (2015). There's also some completely different approach to tackling long-term dependencies, like Clockwork RNNs by Koutnik, et al. (2014). Which of these variants is best? Do the differences matter? Greff, et al. (2015) do a nice comparison of popular variants, finding that they're all about the same. Jozefowicz, et al. (2015) tested more than ten thousand RNN architectures, finding some that worked better than LSTMs on certain tasks.

5.6.1 .4 Conclusion

Earlier, I mentioned the remarkable results people are achieving with RNNs. Essentially all of these are achieved using LSTMs. They really work a lot better for most tasks!

Written down as a set of equations, LSTMs look pretty intimidating. Hopefully, walking through them step by step in this essay has made them a bit more approachable.

LSTMs were a big step in what we can accomplish with RNNs. It's natural to wonder: is there another big step? A common opinion among researchers is: "Yes! There is a next step and it's attention!" The idea is to let every step of an RNN pick information to look at from some larger collection of information. For example, if you are using an RNN to create a caption describing an

image, it might pick a part of the image to look at for every word it outputs. In fact, Xu, *et al.*(2015) do exactly this – it might be a fun starting point if you want to explore attention! There's been a number of really exciting results using attention, and it seems like a lot more are around the corner…

Attention isn't the only exciting thread in RNN research. For example, Grid LSTMs by Kalchbrenner, *et al.* (2015) seem extremely promising. Work using RNNs in generative models – such as Gregor, *et al.* (2015), Chung, *et al.* (2015), or Bayer & Osendorfer (2015) – also seems very interesting. The last few years have been an exciting time for recurrent neural networks, and the coming ones promise to only be more so!

Chapter 6.

Model

In this section we describe our base OBJ2TEXT model for encoding object layouts to produce text (section 6.1), as well as two further variations to use our model to generate captions for real images: OBJ2TEXT-YOLO which uses the YOLO object detector (Redmon and Farhadi, 2017) to generate layouts of object locations from real images (section 6.2), and OBJ2TEXT-YOLO + CNN-RNN which further combines the previous model with an encoder-decoder image captioning which uses a convolutional neural network to encode the image (section 6.3).

6.1 OBJ2TEXT

OBJ2TEXT is a sequence-to-sequence model that encodes an input object layout as a sequence, and decodes a textual description by predicting the next word at each time step. Given a training data set comprising N observations $\{<o^{(n)}, l^{(n)}>\}$, where $\{<o^{(n)}, l^{(n)}>\}$ is a pair of sequences of input category and location vectors, together with a corresponding set of target captions $\{s^{(n)}\}$, the encoder and decoder are trained jointly by minimizing a loss function over the training set using stochastic gradient descent:

Equation 1 $$\boldsymbol{W = arg\,min \sum_{n=1}^{N} L(\{<o(n), l(n)>\}, \{s(n)\})}$$

Where , W1 in W is Encoder parameter and W2 in W is decoder parameter. The loss function is a negative log likelihood function of generated description given the encoded object layout.

Equation 2 $$L(\{<o(n), l(n)>\}, \{s(n)\}) = -\log p(s^n \mid h_L^n, W_2)$$

where h_L^n is computed using the LSTM-based encoder (eqs. 3, and 4) from the object layout inputs $<o(n), l(n)>$ and $p(s^n \mid h_L^n, W_2)$ is computed using the LSTM-based decoder (eqs. 5, 6 and 7). At inference time we encode an input layout <o, l> into its representation h_L, and sample a sentence word by word based on $p(s_t \mid h_L, s_{<t})$ as computed by the decoder in time-step t. Finding the optimal sentence $s = \arg\max_s p(s \mid h_L)$ requires the evaluation of an exponential number of sentences as in each time-step we have K number of choices for a word vocabulary of size K. As a common practice for an approximate solution, we follow (Vinyals et al., 2015) and use beam search to limit the choices for words at each time-step by only using the ones with the highest probabilities.

Encoder: The encoder at each time-step t takes as input a pair $<o_t, l_t>$, where o_t is the object category encoded as a one-hot vector of size V , and $l_t = [B^t_x, B^t_y, B^t_w, B^t_h]$ is the location configuration vector that contains left-most position, topmost position, and the width and height of the bounding box corresponding to object ot, all normalized in the range [0,1] with respect to input image dimensions. o_t and l_t are mapped to vectors with the same size k and added to form the input x_t to one time-step of the LSTM-based encoder as follows:

Equation 3 $$x_t = W_o o_t + (W_l l_t + b_l), \qquad x_t \in R^k$$

in which $W_o \in R^{k \times V}$ is a categorical embedding matrix (the word encoder), and $W_l \in R^{k \times 4}$ and bias $b_l \in R^k$ are parameters of a linear transformation(the object location encoder).

Setting initial value of cell state vector $c^e_o = 0$ and hidden state vector $h_o^e = 0$, the LSTM-based

encoder takes the sequence of input (x_1, x_2 , ...x_{T1}) and generates a sequence of hidden state vectors (h^e_1, …...h^e_{T1}) using the following step function (we omit cell state variables and internal transition gates for simplicity as we use a standard LSTM cell definition):

Equation 4 $\mathbf{h^e_t = LSTM(h^e_{t-1}, x_t; W_1)}$

We use the last hidden state vector $h_L = h^e_{T1}$ as the encoded representation of the input layout $<o_t, l_t>$ to generate the corresponding description s.

Decoder: The decoder takes the encoded layout h_L as input and generates a sequence of multinomial distributions over a vocabulary of words using an LSTM neural language model. The joint probability distribution of generated sentence s = (s_1, s_2 :::; s_{T2}) is factorized into products of conditional probabilities:

Equation 5 $\mathbf{p(s|h_L) = \prod_{t=1}^{T2} p(s_t|h_L, s_{<t})}$

where each factor is computed using a softmax function over the hidden states of the decoder LSTM as follows:

Equation 6 $\mathbf{p(s_t|h_L, s_{<t}) = softmax(W_h| h^d_{t-1} + b_h)}$

Equation 7 $\mathbf{h^d_t = LSTM(h^d_{t-1}, W_s s_t; W2)}$

where W_s is the categorical embedding matrix for the one-hot encoded caption sequence of symbols. By setting $h^d_{-1} = 0$ and $c^d_{-1} = 0$ for the initial hidden state and cell state, the layout representation is encoded into the decoder network at the 0 time step as a regular input:

Equation 8 $\mathbf{h^d_0 = LSTM(h^d_{-1}, h_L; W2)}$

We use beam search to sample from the LSTM as is routinely performed in previous literature in order to generate text.

6.2 OBJ2TEXT-YOLO

For the task of image captioning we propose OBJ2TEXT-YOLO. This model takes an image as input, extracts an object layout (object categories and locations) with a state-of-the-art object detection model YOLO (Redmon and Farhadi, 2017), and uses OBJ2TEXT as described in section 6.1 to generate a natural language description of the input layout and hence, the input image. The model is trained using the standard back-propagation algorithm, but the error is not back-propagated to the object detection module.

6.3 OBJ2TEXT-YOLO + CNN-RNN

For the image captioning task we experiment with a combined model where we take an image as input, and then use two separate computation branches to extract visual feature information and object layout information. These two streams of information are then passed to an LSTM neural language model to generate a description. Visual features are extracted using the VGG-16 (Simonyan and Zisserman, 2015) convolutional neural network pre-trained on the ImageNet classification task (Russakovsky et al., 2015). Object layouts are extracted using the YOLO object detection system and its output object locations are encoded using our proposed OBJ2TEXT encoder. These two streams of information are encoded into vectors of the same size and their sum is input to the language model to generate a textual description. The model is trained using the standard back-propagation algorithm where the error is back-propagated to both branches but not the object detection module. The weights of the image CNN model are fine-tuned only after the layout encoding branch is well trained but no significant overall performance improvements were observed.

Chapter 7

Experimental Setup

We evaluate the proposed models on the MSCOCO (Lin et al., 2014) dataset which is a popular image captioning benchmark that also contains object extent annotations. In the object layout captioning task the model uses the ground truth object extents as input object layouts, while in the image captioning task the model takes raw images as input. The qualities of generated descriptions are evaluated using both human evaluations and automatic metrics. We train and validate our models based on the commonly adopted split regime (113,287 training images, 5000 validation and 5000 test images) used in (Karpathy et al., 2016), and also test our model in the MSCOCO official test benchmark.

We implement our models based on the open source image captioning system Neuraltalk2 (Karpathy et al., 2016). Other configurations including data preprocessing and training hyper-parameters also follow Neuraltalk2. We trained our models using a GTX1080 GPU with 8GB of memory for 400k iterations using a batch size of 16 and an Adam optimizer with alpha of 0.8, beta of 0.999 and epsilon of 1e-08. Descriptions of the CNN-RNN approach are generated using the publicly available code and model checkpoint provided by Neuraltalk2 (Karpathy et al., 2016). Captions for online test set evaluations are generated using beam search of size 2, but score histories on split validation set are based on captions generated without beam search (i.e. max sampling at each time-step).

Ablation on Object Locations and Counts:

We setup an experiment where we remove the input locations from the OBJ2TEXT encoder to study the effects on the generated captions, and confirm whether the model is actually using spatial information during surface realization. In this restricted version of our model the LSTM encoder at each time step only takes the object category embedding vector as input. The OBJ2TEXT model additionally encodes different instances of the same object category in different time steps, potentially encoding in some of its hidden states information about how many objects of a particular class are in the image. For example, in the object annotation presented in the input in Figure 1, there are two instances of "person". We perform an additional experiment where our model does not have access neither to object locations, nor the number of object instances by providing only a set of object categories. Note that in this set of experiments the object layouts are given as inputs, thus we assume full access to ground-truth object annotations, even in the test split. In the experimental results section we use the "-GT" postfix to indicate that input object layouts are obtained from ground-truth object annotations provided by the MS-COCO dataset.

Image Captioning Experiment:

In this experiment we assess whether the image captioning model OBJ2TEXT-YOLO that only relies on object categories and locations could give comparable performance with a CNN-RNN model based on Neuraltalk2 (Karpathy et al., 2016) that has full access to visual image features. We also explore how much does a combined OBJ2TEXT-YOLO + CNN-RNN model could improve over a CNN-RNN model by fusing object counts and location information that is not explicitly encoded in a traditional CNN-RNN approach.

Human Evaluation Protocol.

We use a two alternative forced-choice evaluation (2AFC) approach to compare two methods that generate captions. For this, we setup a task on Amazon Mechanical Turk where users are presented with an image and two alternative captions, and they have to choose the caption that best describes the image. Users are not prompted to use any single criteria but rather a holistic assessment of the captions, including their semantics, syntax, and the degree to which they describe the image content. In our experiment we randomly sample 500 captions generated by various models for MS COCO online test set images, and use three users per image to obtain annotations. Note that three users choosing randomly between two options have a chance of 25% to select the same caption for a given image. In our experiments comparing method A vs method B, we report the percentage of times A was picked over B (Choice-all), the percentage of times all users selected the same method, either A or B, (Agreement), and the percentage of times A was picked over B only for these cases where all users agreed (Choice-agreement).

Chapter 8

Results

Impact of Object Locations and Counts: Figure 3a shows the CIDEr (Vedantam et al., 2015a), and BLEU-4 (Papineni et al., 2002) score history on our validation set during 400k iterations of training of OBJ2TEXT, as well as a version of our model that does not use object locations, and a version of our model that does not use neither object locations nor object counts. These results show that our model is effectively using both object locations and counts to generate better captions, and absence of any one of these two cues affects performance.

Table 1 confirms these results on the test split after a full round of training. Furthermore, human evaluation results in the first row of Table 2 show that the OBJ2TEXT model with access to object locations is preferred by users, especially in cases where all evaluators agreed on their choice (62% over the baseline that does not have access to locations). In Figure 4 we additionally present qualitative examples showing predictions side-by-side between OBJ2TEXT-GT and OBJ2TEXT-GT (no obj-locations). These results indicate that 1) perhaps not surprisingly, object counts is useful for generating better quality descriptions, and 2) object location information when properly encoded, is an important cue for generating more accurate descriptions. We additionally implemented a nearest neighbor baseline by representing the objects in the input layout using an orderless bag-of-words representation of object counts and the CIDEr score on the test split was only 0.387.

On top of OBJ2TEXT we additionally experimented with the global attention model proposed in (Luong et al., 2015) so that a weighted combination of the encoder hidden states are forwarded

to the decoding neural language model, however we did not notice any overall gains in terms of accuracy from this formulation. We observed that this model provided gains only for larger input sequences where it is more likely that the LSTM network forgets its past history (Bahdanau et al., 2015). However in MS-COCO the average number of objects in each image is rather modest, so the last hidden state can capture well the overall nuances of the visual input.

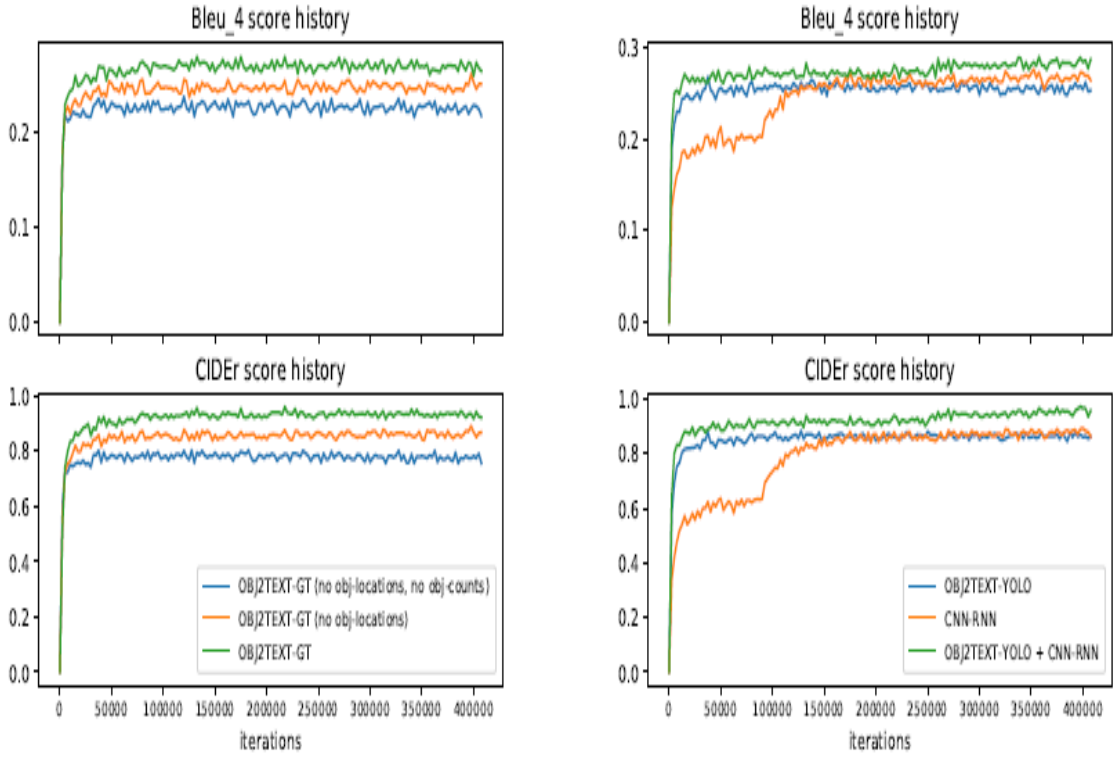
Object Layout Encoding for Image Captioning:

Figure 3b shows the CIDEr, and BLEU-4 score history on the validation set during 400k iterations of training of OBJ2TEXT-YOLO, CNN-RNN, and their combination. These results show that OBJ2TEXT-YOLO performs surprisingly close to CNN-RNN, and the model resulting from combining the two, clearly outperforms each method alone. Table 3 shows MS-COCO evaluation results on the test set using their online benchmark service, and confirms results obtained in the validation split, where CNN-RNN seems to have only a slight edge over OBJ2TEXT-YOLO which lacks access to pixel data after the object detection stage.

Human evaluation results in Table 2 rows 2, and 3, further confirm these findings. These results show that meaningful descriptions could be generated solely based on object categories and locations information, even without access to color and texture input. The combined model performs better than the two models, improving the CIDEr score of the basic CNN-RNN model from 0.863 to 0.950, and human evaluation results show that the combined model is preferred over the basic CNN-RNN model for 65.3% of the images for which all evaluators were in agreement about the selected method. These results show that explicitly encoded object counts and location information, which is often overlooked in traditional image captioning approaches, could boost the performance of existing models. Intuitively, object layout and visual features are

complementary: neural network models for visual feature extraction are trained on a classification task where object-level information such as number of instances and locations are ignored in the objective. Object layouts on the other hand, contain categories and their bounding-boxes but don't have access to rich image features such as image background, object attributes and objects with categories not present in the object detection vocabulary. Figure 5 provides a three-way comparison of captions generated by the three image captioning models, with preferred captions by human evaluators annotated in bold text. Analysis on actual outputs gives us insights into the benefits of combing object layout information and visual features obtained using a CNN. Our OBJ2TEXT-YOLO model makes many mistakes because of lack of image context information since it only has access to object layout, while CNN-RNN makes many mistakes because the visual recognition model is imperfect at predicting the correct content. The combined model is usually able to generate more accurate and comprehensive descriptions. In this work we only explored encoding spatial information with object labels, but object labels could be readily augmented with rich semantic features that are more detailed descriptions of objects or image regions. For example, the work of You et al. (2016) and Yao et al. (2016b) showed that visual features trained with semantic concepts (text entities mentioned in captions) instead of object labels is useful for image captioning, although they didn't consider encoding semantic concepts with spatial information. In case of object annotations the MS-COCO dataset only provides object labels and bounding-boxes, but there are other datasets such as Flick30K Entities (Plummer et al., 2015), and the Visual Genome dataset (Krishna et al., 2017) that provide richer region-tophrase correspondence annotations. In addition, the fusion of object counts and spatial information with CNN visual features could in principle

benefit other vision and language tasks such as visual question answering. We leave these possible extensions as future work.



(a) Score histories of lesioned versions of the proposed model for the task of object layout captioning.

(b) Score histories of image captioning models. Performance boosts of CNN-RNN and combined model around iteration 100K and 250K are due to fine-tuning of the image CNN model.

Figure 8.1Score histories of various models on the MS COCO split validation set.

Method	Bleu_4	CIDEr	METEOR	ROUGE-L
OBJ2TEXT-GT (no obj-locations, counts)	0.21	0.759	0.215	0.464
OBJ2TEXT-GT (no obj-locations)	0.233	0.837	0.222	0.482
OBJ2TEXT-GT	**0.253**	**0.922**	**0.238**	**0.507**

Table 1: Performance of lesioned versions of the proposed model on the MS COCO split test set.

Alternatives	Choice-all	Choice-agreement	Agreement
OBJ2TEXT-GT vs. OBJ2TEXT-GT (no obj-locations)	54.1%	62.1%	40.6%
OBJ2TEXT-YOLO vs. CNN+RNN	45.6%	40.6%	54.7%
OBJ2TEXT-YOLO + CNN-RNN vs. CNN-RNN	58.1%	65.3%	49.5%
OBJ2TEXT-GT vs. HUMAN	23.6%	9.9%	58.8%

Table 2: Human evaluation results using two-alternative forced choice evaluation. Choice-all is percentage the first alternative was chosen. Choice-agreement is percentage the first alternative was chosen only when all annotators agreed. Agreement is percentage where all annotators agreed (random is 25%).

MS COCO Test Set Performance	CIDEr	ROUGE-L	METEOR	B-4	B-3	B-2	B-1
5-Refs							
OBJ2TEXT-YOLO	0.830	0.497	0.228	0.262	0.361	0.500	0.681
CNN-RNN	0.857	0.514	0.237	0.283	0.387	0.529	0.705
OBJ2TEXT-YOLO + CNN-RNN	**0.932**	**0.528**	**0.250**	**0.300**	**0.404**	**0.546**	**0.719**
40-Refs							
OBJ2TEXT-YOLO	0.853	0.636	0.305	0.508	0.624	0.746	0.858
CNN-RNN	0.863	0.654	0.318	0.540	0.656	0.775	0.877
OBJ2TEXT-YOLO + CNN-RNN	**0.950**	**0.671**	**0.334**	**0.569**	**0.686**	**0.802**	**0.896**

Table 3: The 5-Refs and 40-Refs performances of OBJ2TEXT-YOLO, CNN-RNN and the combined approach on the MS COCO online test set. The 5-Refs performance is measured using 5 ground-truth reference captions, while 40-Refs performance is measured using 40 ground-truth reference captions.

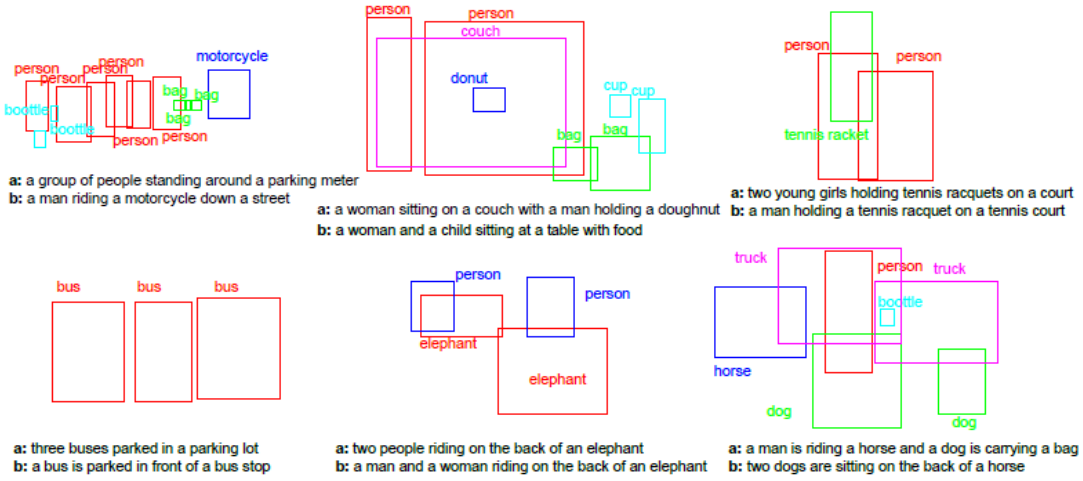


Figure 8.2 Qualitative examples comparing generated captions of (**a**) OBJ2TEXT-GT, and (**b**) OBJ2TEXT-GT (no obj-locations).

Chapter 9

Conclusion

We introduced OBJ2TEXT, a sequence-to-sequence model to generate visual descriptions for object layouts where only categories and locations are specified. Our proposed model shows that an orderless visual input representation of concepts is not enough to produce good descriptions, but object extents, locations, and object counts, all contribute to generate more accurate image descriptions. Crucially we show that our encoding mechanism is able to capture useful spatial information using an LSTM network to produce image descriptions, even when the input is provided as a sequence rather than as an explicit 2D representation of objects. Additionally, using our proposed OBJ2TEXT model in combination with an existing image captioning model and a robust object detector we showed improved results in the task of image captioning.

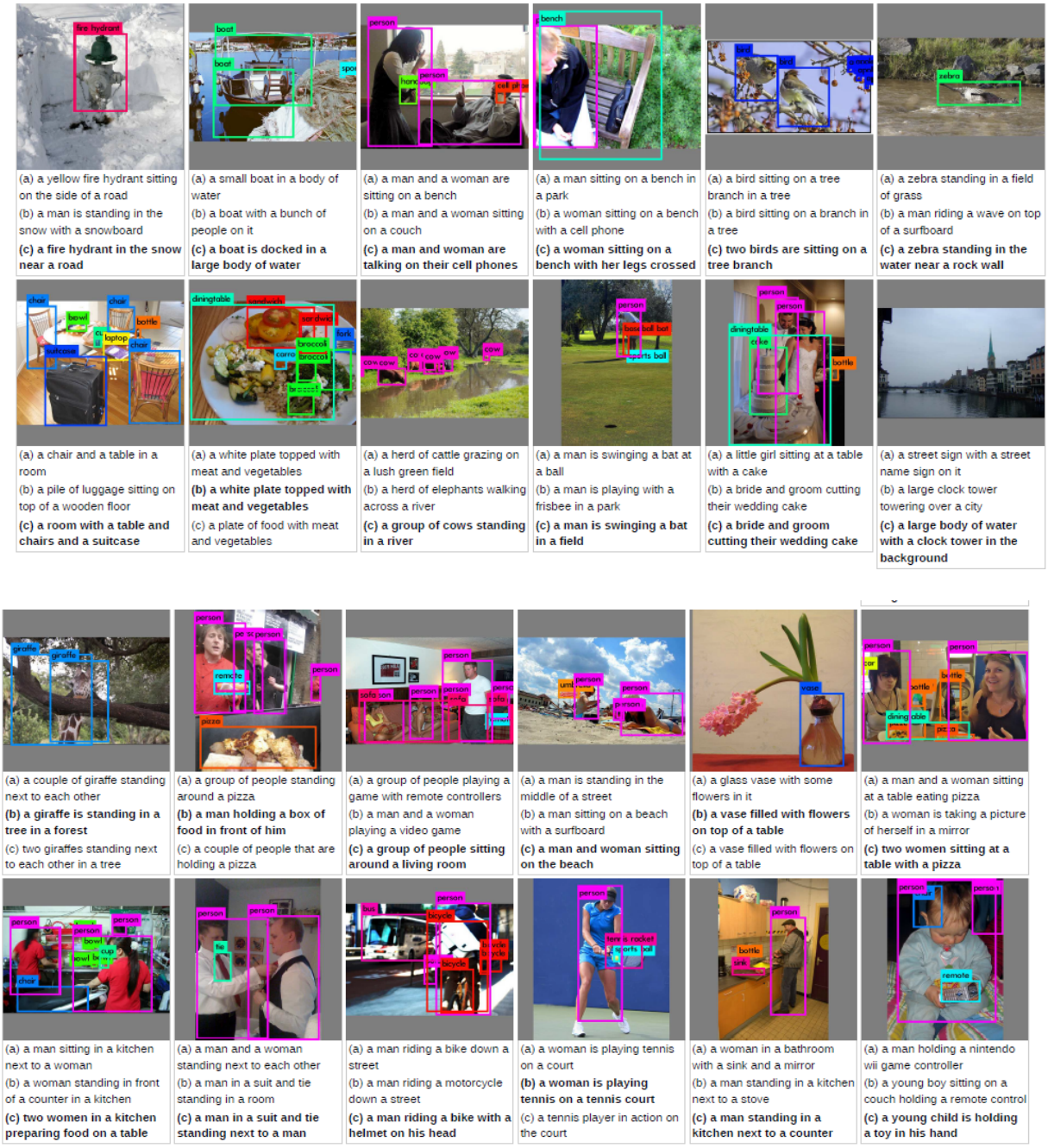


Figure 9.1 Qualitative examples comparing the generated captions of (a) OBJ2TEXT-YOLO, (b) CNN-RNN and (c) OBJ2TEXT-YOLO + CNN-RNN. Images are selected from the 500 human evaluation images and annotated with YOLO object detection results. Captions preferred by human evaluators with agreement are highlighted in bold text.

References

[1] Hours of video uploaded to YouTube every minute as of July 2015, Accessed June 23, 2016. http://www.statista.com/statistics/259477/ hours-of-video-uploaded-to-youtube-every-minute/.
[2] Instagram Statistics, Accessed June 23, 2016. https://www.instagram.com/press.
[3] Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. arXiv abs/1409.0473 (2014).
[4] Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I. J., Bergeron, A., Bouchard, N., and Bengio, Y. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.
[5] Berger, A. L., Pietra, V. J. D., and Pietra, S. A. D. A maximum entropy approach to natural language processing. Computational Linguistics 22 (1996).
[6] Brown, P. F., deSouza, P. V., Mercer, R. L., Pietra, V. J. D., and Lai, J. C. Class-based n-gram models of natural language. Computational Linguistics 18 (1992).
[7] Chang, C.-C., and Lin, C.-J. LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology (2011).
[8] Chen, D. L., and Dolan, W. B. Collecting highly parallel datafor paraphrase evaluation. In Proceedings of the ACL Conference on Human Language Technologies (2011), Association for Computational Linguistics.
[9] Chen, X., Hao Fang, T.-Y. L., Vedantam, R., Gupta, S.,Doll~A_r, P., and Zitnick, C. L. Microsoft COCO captions: Data collection and evaluation server. arXiv abs/1504.00325 (2015).
[10] Cortes, C., and Vapnik, V. Support-vector networks. Machine Learning 20 (1995).
[11] Cui, Y., Ruggero Ronchi, M., and Lin, T.-Y. 1st captioning challenge slides. Large-scale Scene Understanding Workshop, 2015.
[12] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., and Fei-Fei, L. ImageNet: A large-scale hierarchical image database. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2009).
[13] Denkowski, M., and Lavie, A. Meteor universal: Language speci_c translation evaluation for any target language. In Proceedings of the Ninth Workshop on Statistical Machine Translation (2014), ACL.
[14] Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., and Darrell, T. Long-term recurrent convolutional networks for visual recognition and description. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2015).
[15] Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. DeCAF: A deep convolutional activation feature for generic visual recognition. In Proceedings of the International Conference on Machine Learning (2014).
[16] Fang, H., Gupta, S., Iandola, F., Srivastava, R., Deng, L., Dollar, P., Gao, J., He, X., Mitchell, M., Platt, J., Zitnick, L., and Zweig, G. From captions to visual concepts and back. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2015).
[17] Farhadi, A., Hejrati, M., Sadeghi, M. A., Young, P., Rashtchian, C., Hockenmaier, J., and Forsyth, D. Every picture tells a story: Generating sentences from images. In European Conference on Computer Vision (2010).

[18] Frome, A., Corrado, G. S., Shlens, J., Bengio, S., Dean, J., Mikolov, T., et al. Devise: A deep visual-semantic embedding model. In Advances in neural information processing systems (2013).

[19] Gong, Y., Wang, L., Guo, R., and Lazebnik, S. Multi-scale orderless pooling of deep convolutional activation features. arXiv abs/1403.1840 (2014).

[20] He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectiers: Surpassing human-level performance on imagenet classi_cation. In Proceedings of the IEEE International Conference on Computer Vision (2015).

[21] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In Proceeding of IEEE Conference on Computer Vision and Pattern Recognition (2016).

[22] Hinton, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[23] Hochreiter, S., and Schmidhuber, J. Long short-term memory. Neural Computing 9 (1997).

[24] Hodosh, M., Young, P., and Hockenmaier, J. Framing image description as a ranking task: Data, models and evaluation metrics. Journal of Arti_cial Intelligence Research 47 (2013).

[25] Ji, S., Xu, W., Yang, M., and Yu, K. 3d convolutional neural networks for human action recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence (2013).

[26] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J.Girshick, R., Guadarrama, S., and Darrell, T. Ca_e: Convolutional architecture for fast feature embedding. arXiv abs/1408.5093 (2014).

[27] Johnson, J., Karpathy, A., and Fei-Fei, L. Densecap: Fully convolutional localization networks for dense captioning. arXiv abs/1511.07571 (2015).

[28] Johnson, J., Krishna, R., Stark, M., Li, L.-J., Shamma, D. A., Bernstein, M. S., and Fei-Fei, L. Image retrieval using scene graphs. In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (2015), IEEE.

[29] Karpathy, A., and Fei-Fei, L. Deep visual-semantic alignments for generating image descriptions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2015).

[30] Karpathy, A., Joulin, A., and Fei-Fei, L. Deep fragment embeddings for bidirectional image sentence mapping. In Advances in neural information processing systems (2014).

[31] Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., and Fei-Fei, L. Large-scale video classi_cation with convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2014).

[32] Kim, Y. Convolutional neural networks for sentence classi_cation. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (2014), Association for Computational Linguistics.

[33] Krishna, R., Zhu, Y., Groth, O., Johnson, J., Hata, K., Kravitz, J., Chen, S., Kalantidis, Y., Li, L.-J., Shamma, D. A., Bernstein, M., and Fei-Fei, L. Visual genome: Connecting language and vision using crowdsourced dense image annotations. arXiv abs/1602.07332 (2016).

[34] Krizhevsky, A., Sutskever, I., and Hinton, G. ImageNet classificcation with deep convolutional neural networks. In Advances in neural information processing systems (2012).

[35] Kulkarni, G., Premraj, V., Ordonez, V., Dhar, S., Li, S., Choi, Y., Berg, A. C., and Berg, T. L. Babytalk: Understanding and generating simple image descriptions. IEEE Transactions on Pattern Analysis and Machine Intelligence 35 (2013).