IMPROVING IMAGE CAPTIONING WITH LANGUAGE MODELING REGULARIZATIONS

by

Okan Ulusoy

B.S., Electrical and Electronics Engineering, Boğaziçi University, 2015

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Electrical and Electronics Engineering
Boğaziçi University

2019

ACKNOWLEDGEMENTS

I must thank for many people contributing to this thesis during three years I have spent as a MS student. Above all, I have to thank profoundly my thesis advisors Prof. Emin Anarım and Dr. Ceyhun Burak Akgül though I cannot be thankful enough.

I must thank Prof. Emin Anarım for his great support, guidance, patience and insightful comments during the last three years. He was a role model for me in many respects with his ideals, foresight and passion for his work. He is one of the most genuine, caring people I have ever met. I am forever grateful that he forgave me even when I made a big mistake. I have a lot of respect and admiration for Prof. Emin Anarım and I am very thankful to him for accepting to become my thesis advisor.

I must thank Dr. Ceyhun Burak Akgül who, through many emails, meetings and discussions, paved the path for me to become a competent researcher. He rekindled my passion for computer vision with not only his teachings but also with his ambitions. He gave me guidance on both academic and non-academic topics, when I needed it the most. He is a remarkably smart and supportive person and I think that is why I have never hesitated to share my problems with him. I am very thankful to Dr. Ceyhun Burak Akgül for not only teaching me the way to think about a problem but also how to communicate more clearly with others.

I have to thank my thesis jury members Prof. Dr. Tayfun Akgül and Prof. Dr. Burak Acar for their time and for providing invaluable advice regarding my thesis.

I must thank my sister and my parents for always believing and supporting me.

I would also like thank my dear friends, Ersoy Çalışkan, Miraç Göksu Öztürk, Cem Güray, Mehmet Soydaş, Yasin Özkanca and Cansu Zeybekgil.

I am also thankful to Arçelik, Burgan Bank and Özyeğin University for giving me permission for attending my graduation studies whenever I needed. From these companies I would especially like to thank Cenk Demiroğlu, Salih Sertbaş, Uğur Halatoğlu and Ramazan Işık.

ABSTRACT

IMPROVING IMAGE CAPTIONING WITH LANGUAGE MODELING REGULARIZATIONS

Inspired by the recent work in language modeling, we investigate the effects of a set of regularization techniques on the performance of a recurrent neural network based image captioning model. Using these techniques, we achieve 13 Bleu-4 points improvements over using no regularizations. We show that our model does not suffer from loss-evaluation mismatch and also connect the model performance to dataset properties by running experiments on MSCOCO dataset. Further, we propose two different applications for our image captioning model, namely human in the loop system and zero shot object detection. The former application further improves CIDEr score of our best model by 30 points using only the first two tokens of a reference sentence of an image. In the latter one, we train our image captioning model as an object detector which classifies each objects in an image without finding their location. The main advantage of this detector is that it does not require object locations during the training phase.

ÖZET

TANIM OLUŞTURMA MODELİNİ DİL MODELLEMEDEKİ BAŞARIM İYİLEŞTİRME TEKNİKLERİ İLE GELİŞTİRMEK

Dil modelleme konusundaki son çalışmalardan esinlenerek, bir takım başarım iyileştirme tekniğinin tekrarlayan bir sinir ağına dayalı bir tanım oluşturma modelinin performansı üzerindeki etkilerini araştırdık. Bu teknikleri kullanarak, hiçbir iyileştirme tekniği kullanmamaya oranla performansımızda 13 Bleu-4 puan iyileştirdik. Modelimizde hata-değerlendirme uyumsuzluğu olmadığını MSCOCO veri setinde deneyler yaparak gösterdik. Ayrıca, model performansının veri kümesin özelliklerine bağladığını gösterdik. Ek olarak, tanım oluşturma modelimizi temel alarak insan-bilgisayar hibrid tanım oluşturma modeli ve tek seferde nesne tanıma modeli isimlerinde iki farklı uygulama geliştirdik. İlk uygulama ile en iyi modelimizin CIDEr puanını bir görüntünün referans cümlesinin yalnızca ilk iki kelimesini kullanarak 30 puan arttırdık. İkinci uygulamamızda, tanım oluşturma modelimizi bir resimdeki nesnelerin konumlarını bulamadan sınıflandıran bir nesne dedektörü olarak eğittik. Bu dedektörün temel avantajı, eğitim aşaması sırasında nesne konumlarını gerektirmemesidir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$\boldsymbol{x}$	A vector
$\boldsymbol{X}$	A matrix
$\mathcal{X}$	A sequence
x_i	i-th element of $\mathcal{X}$ in which each element in the sequence is a scalar
$\boldsymbol{x_1}$	i-th element of $\mathcal{X}$ in which each element in the sequence is a vector
$\boldsymbol{X}[i,j]$	an element on the i^{th} row and j^{th} column of a two dimensional matrix $\boldsymbol{X}$
$\boldsymbol{X}[i,:]$	i^{th} row of $\boldsymbol{X}$
$\boldsymbol{X}[:,j]$	j^{th} column of $\boldsymbol{X}$
$\boldsymbol{x}[i]$	i^{th} element of vector $\boldsymbol{x}$
$\boldsymbol{X}[i,:,:]$	i^{th} row of matrix $\boldsymbol{X}$
$\boldsymbol{I}$	A WxHxN dimensional image ,where W, H corresponds to width and height of the image whereas N is the number of color channels of the image (e.g. $N = 1$ for gray scale images, $N = 3$ for RGB images)

LIST OF ACRONYMS/ABBREVIATIONS

BLEU	Bilingual Evaluation Understudy
BRNN	Bidirectional Recurrent Neural Network
CONV	Convolutional Layer
ELU	Exponential Linear Units
EOS	End of Sentence Token
FC	Fully Connected Layer
TN	False Negative
TP	False Positive
CIDEr	Consensus-based Image Description Evaluation
CNN	Convolutional Neural Network
GPU	Graphics Processing Unit
IDF	Inverse Document Frequency
ILSVRC	Imagenet Large Scale Visual Recognition Challenge
LBLM	Log-Bilinear Language Model
LCS	Largest Common Subsequence
LReLU	Leaky Rectified Linear Unit
LSTM	Long Short Term Memory
METEOR	Metric for Evaluation of Translation with Explicit Ordering
MS COCO	Microsoft Common Objects in Context
PReLU	Parametric Rectified Linear Unit
RCNN	Regional Convolutional Neural Network
REC	Recurrent Layer
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
ROGUE	Recall-Oriented Understudy for Gisting Evaluation
TF	Term Frequency
TN	True Negative
TP	True Positive

SOS	Start of Sentence Token
UNK	Unknown Token

1. INTRODUCTION

1.1. Problem Overview

Visual descriptions constitute the core of cognitive process involved in understanding visual scenes. Humans can describe these scenes to one another without any difficulty thanks to their ability to make connections between visual and linguistic worlds. However, modeling this task is challenging for computers for two main reasons [1–3]:

(i) The model should be rich enough to detect objects, which is important to create visual descriptions. Since the generated description could change significantly from one person to another, datasets with multiple sentence annotations per image is needed to determine important objects.

(ii) Language modeling is also necessary for describing the objects and their relationship with one another in a language like English. The language model should be capable of generating meaningful descriptions through the inferred relations between words and detected objects.

Image classification addresses mainly the first part of this challenge. Russakovsky *et al.* [4] reported that the accuracy of image classification has increased with the help of ImageNet dataset and annual Imagenet Large Scale Visual Recognition Challenge (ILSVRC) from 2010 to 2015. Krizhevsky *et al.* [5] introduced AlexNet, a type of Convolutional Neural Network (CNN) [6], and achieved the error rate of 15.3%, which is 10.9% better than the second-best entry on ILSVRC-2012. Afterwards, various forms of CNNs are used in image classification (e.g. VGGNet [7], GoogleNet [8] and ResNet [9]) and the error rate has decreased significantly.

The second part of the challenge consists of creating word representations. Each representation should include both the desired word and its connections with other words in the sentence. Recurrent Neural Networks (RNNs) [10], Bidirectional Recur-

rent Neural Network [11], biterm [12] and dependency trees [13] could be used to create these descriptions.

The generation of descriptions for visual scenes has attracted many researchers over the last few years. Farhadi *et al.* [14] used a retrieval-based approach where a test image is described with the most similar sentence in the training set. Li *et al.* [15] suggested a similar method and combined the different parts of training sentences. Gupta and Mannem [16] proposed a template-based method and identified the components from the image. Several multimodal methods are also designed for both multidirectional image-sentence retrieval [17] and unique description generation [1, 3, 18].

1.2. Applications

Automatically generating image descriptions could help visually impaired people better understand the world around them. These descriptions could also be used to support children in their education. Moreover, once an image is automatically translated into a textual description, it could be used as a query for text-based search engines. Last but not least, a human in the loop description generation system, where a person creates a few words and the description generation system produces the rest of it, could be envisioned to increase the accuracy of the generated descriptions on one hand and to facilitate image annotation tasks on the other. People with speech disorders could use these systems to improve their life standards. Hybrid systems could also be used to shorten the decision-making processes for a system where a human interpretation of a visual scene is necessary, such as surveillance.

1.3. Related Works

1.3.1. Retrieval Based Methods

Farhadi *et al.* [14] suggested that the relation between images (visual scenes) and sentences (their descriptors) could be represented in meaning space with a triplet of <object, action, scene>. Each image and sentence is mapped to this space. A high score between an image and a sentence is obtained when these meanings were similar. One limitation of this method is that each component of the triplet could take a value from a finite set. Moreover, this method creates descriptions by retrieving sentences from the training set instead of generating unique sentences. These problems significantly restrict the capacity of the model.

Li *et al.* [15] proposed an approach for generating image descriptions using web-scale n-grams, which provides the frequency count of word n-grams from Web pages [19]. Each image is represented with a set of triplets <objects, attributes, spatial relationship>, similar to the work of Farhadi *et al.* [14]. Image features are used to determine these triplets. Afterwards, candidate sentences, which contains similar triplets, are collected from web scale n-gram. Descriptor generation is achieved by combining some parts of candidate sentences. Although unique sentences are generated with this method, candidate sentences limit the uniqueness of the descriptions.

1.3.2. Template Based Methods

Gupta and Mannem [16] generated image descriptions from the image annotation, which is described as the labeling of images. They showed that a simple description could be described with a set of fundamental elements, namely objects, attributes, attribute-object pair, subject, verb and proposition. For example, "A brown dog is sitting next to a white wall" description was represented with two objects, "dog" and "wall", two attributes, "brown" and "white", two attribute-object pair, "brown dog" and "white wall", one subject, "dog", one verb, "sit" and one proposition, "next to", in their work. Their model has produced descriptions with only two known fundamental

elements, namely objects and attributes. Other elements are predicted from the training images. Although meaningful descriptions are generated, the sentence structure was hand-designed. Description structure should be free of assumptions to generate unique and complex sentences.

1.3.3. Multimodal Methods

Baltrušaitis *et al.* [20] defines modality as "the way in which something happens or is experienced". Multimodal methods refer to the models which uses multiple such modalities (e.g. images, sound or text) [21]. Baltrušaitis *et al.* [20] suggested that humans could understand the natural phenomena better by understanding the connections between different modalities. Frome *et al.* [22] demonstrated the power of multimodal machine learning in visual recognition by connecting image domain to text domain. Each word in the text domain was transformed to an embedding vector via skip-gram text model, which maps the semantically related words close to each other. Image features were extracted with AlexNet [5] and mapped to vector embedding space via a fully connected layer. Then, both models are trained with an objective function which favors the equivalence of transformed image features and the embedding vector representation of related image label (i.e. text). The authors have showed that their model correctly classified images even when their image labels were not seen during training.

Karpathy *et al.* [17] proposed to use multimodal methods for the bidirectional retrieval of sentences and images. The retrieval problem could be stated as finding the most relevant images for a given sentence and vice versa. The authors have showed that the performance of the retrieval method has increased after the latent alignment between sentence fragments, which represent the relation between two or more words in a sentence, and image fragments, which contain the whole image and each object in that image separately, are modeled. Each sentence is represented with multiple fragments that are extracted with the typed dependency parser without tree structure, which includes 48 grammatical relations [13]. Object fragments are detected with a Regional Convolutional Neural Network (RCNN) [23]. Afterwards, the whole model

is trained with an objective function which favors both the alignment of true image-sentence fragments and the alignment of true image-sentence pair. The result of their method is interpretable thanks to this image-sentence fragment alignment objective. However, the simplicity induced by discarding tree structure also caused misalignment of some relations.

Kiros *et al.* [18] used an image-text multimodal neural language model for description generation. A modified log-bilinear language model (LBLM) [24] is used to generate the conditional probability of a word in the sentence given all previous words and the corresponding image for that sentence. Each word in the sentence is used as input for the model, whereas extracted image features are used as either bias or for gating. The model was able to predict the most likely word, which should follow a given word sequence. Image descriptions are created with extending the given word sequence with the predicted word and repeating this process as long as desired. Although the generated descriptions gave general information related to the image, the model had two main drawbacks. Firstly, the variety of descriptions is limited by the initial word sequences, since these sequences are not learned by the model but manually chosen. Secondly, a word is related not only with the previous word sequence but also with the following word sequence. For example, an adjective is usually followed by a noun in a sentence. Their model ignored the influence of following word sequence on a word.

Karpathy and Fei-Fei [1] have shown that the multimodal relation between visual and language data could be used to create image descriptions. Initially the latent relation between words and image regions are learned, similar to the work of Karpathy *et al.* [17]. Afterwards, these aligned image regions and words are used to train a multimodal neural network for description generation. In the first part, RCNN [23] is used to extract 19 top scoring image regions, each of which could contain one of the 200 object classes from the detection task of ILSVRC [25]. Word representations are generated with a Bidirectional Recurrent Neural Network (BRNN) [11]. The power of BRNN comes from creating a word representation, which contains information about not only with that word but also with the other words around it. Then, word representations are aligned with a set of image regions and whole image using an alignment objective

similar to the work of Karpathy *et al.* [17]. In the second part, a Multimodal RNN is used to generate the image description. Unlike Kiros *et al.* [18], the authors have used special tokens to indicate the beginning and ending of the description generation process. The most likely word sequence for the description is found via beam search [26]. Their experiment showed that unique image descriptions could be generated with a Multimodal RNN. Furthermore, the performance of their method has increased using the corresponding image region-word pairs inferred by their alignment model.

Vinyals *et al.* [3] also used a multimodal method for description generation similar to Karpathy and Fei-Fei [1]. Descriptions are generated with a Long Short Term Memory (LSTM) network [27] which uses the image features as input at the first-time step. Although the researchers did not use any alignment information, their model outperformed the model of Karpathy and Fei-Fei [1].

1.3.4. Human in the Loop Methods

Human-computer hybrid systems have become popular in the last few years. Holub *et al.* [28] proposed a human-computer hybrid technique to minimize the total number of images that need to be labeled in order to achieve near-optimal performance for image classification. In their work, a human labeled a set of images sequentially. At each iteration, the expected entropy decrease introduced by labeling each image is calculated and the image resulting in the highest decrease in entropy is labeled. They showed that the total number of training images required to achieve near-maximal performance with their method is less than the half of the total number of training images required to achieve similar performances with randomly selecting images for a human to label for text-based web image searches. In some cases, their method achieved similar performances with less than one-tenth of total number of training images.

Branson *et al.* [29] designed a hybrid human-computer system for image classification. In their system, humans have provided some low-level visual attributes which are hard for computers to extract. Afterwards the class of a given object is determined with the help of these attributes. They showed that their hybrid human-computer system could provide up to 66% classification accuracy in comparison to 19% accuracy obtained by only a computer vision system. Similar human-in-the loop systems could be used to increase the accuracy of model generated descriptions for image captioning.

1.4. Contributions and Outline

In this study, we investigate the effects of a set of regularization techniques on the performance of an description generation model using the recent advances in language modeling [30, 31]. We aim to empirically show that the improvements achieved with these techniques are not only specific to language modeling, but also transfers to image captioning task as well. Similar to model proposed by Vinyals *et al.* [3], we use a convolution neural network (CNN) to extract visual features from images and and a long short-term memory network (LSTM) to generate descriptions using these features. Using a set of regularization techniques improves our results up to 13 Bleu-4 points over using no regularizations. Further, we analyze the relationship between data set properties and the model performance. We conclude by introducing two different applications for our image captioning model, namely human in the loop system and zero shot object detection. The former application further improves CIDEr score of our best model by 30 points using only the first two tokens of a reference sentence of an image. In the latter one, we train our image captioning model as an object detector which classifies each objects in an image without finding their location. The main advantage of this detector is that it does not require object locations during the training phase. We obtain up to 0.99 precision and 0.86 recall on the validation set of retail dataset, which is described in Section 4.1, with our object detector.

2. MATHEMATICAL BACKGROUND

This dissertation builds upon the fundamentals of machine learning. This section presents the basic principles of machine learning and some of the main models used in our image description generation architecture.

Through this section, we use small case letters to represent scalars (e.g., x, y), bold small case ones to represent vectors (e.g., $\boldsymbol{x}, \boldsymbol{y}$), bold upper case ones to represent matrices (e.g., $\boldsymbol{X}, \boldsymbol{Y}$), calligraphic upper case ones to represent sequences (e.g., $\mathcal{X}, \mathcal{Y}$). If each element in the sequence is a scalar, small case letters with underscore are used (e.g., $x_1, x_2, \ldots, x_N$); if each element is a vector, bold small case letters are used ($\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{\ldots}, \boldsymbol{x_N}$) and so on. Additionally, an element on the i^{th} row and j^{th} column of a two dimensional matrix $\boldsymbol{X}$ is represented with $\boldsymbol{X}[i, j]$, i^{th} row of $\boldsymbol{X}$ is represented with $\boldsymbol{X}[i, :]$ and j^{th} column of $\boldsymbol{X}$ is represented with $\boldsymbol{X}[:, j]$. Similar notations are also used for vectors (e.g. $\boldsymbol{x}[i]$ for the element on i^{th} row) and matrices with more than two dimensions (e.g. $\boldsymbol{X}[i, :, :]$ for the matrix on i^{th} row). In some cases, we have used these notations to represent some other concepts, which could be inferred from the context.

2.1. Basics of Machine Learning

Machine learning is used for the extraction of latent information between an input space and an output space when the humans are not able to obtain the exact formula for this mapping or when the mapping is computationally expensive [32]. Let us assume that x_1 and x_2 constitute the input space, y forms the output space and $g(\ldots)$ is the real mapping from the input space to the output space so that $y = g(x_1, x_2)$. If Equation (2.1) is valid, then we can use the summation for the mapping.

$$\forall (x_1, x_2) \in \mathbf{R^2} : \qquad g(x_1, x_2) = x_1 + x_2 \tag{2.1}$$

Let us consider another case such that the input space consists of a set of images and we want to map each image to a description. For example, the image in Figure 2.1 could be mapped to *"There is a cat."* sentence. Although this task is trivial for us, it is impossible to represent this mapping with a deterministic function.

We could represent the cat image with $\boldsymbol{I}$, a matrix of size WxHxN where W, H corresponds to width and height of the image whereas N is the number of color channels of the image (e.g. $N = 1$ for gray scale images, $N = 3$ for RGB images). We could also represent each word or alphanumeric symbol in the description with a vector using either *one-hot encoding* or *variable-sized encoding* (explained in Section 2.2.5) so that we could formulate our problem. Following the notation in Equation (2.2), the description could be represented as $\mathcal{Y} = \{\boldsymbol{y_1}, \boldsymbol{y_2}, \boldsymbol{y_3}, \boldsymbol{y_4}, \boldsymbol{y_5}\}$. Our objective is to find the function $g(\boldsymbol{I}) = \mathcal{Y}$

$$\underbrace{There}_{\boldsymbol{y_1}}\ \underbrace{is}_{\boldsymbol{y_2}}\ \underbrace{a}_{\boldsymbol{y_3}}\ \underbrace{cat}_{\boldsymbol{y_4}}\ \underbrace{.}_{\boldsymbol{y_5}} \tag{2.2}$$

Figure 2.1. An image with description " *There is a cat.*"

Let $f(\ldots)$ represent the predicted mapping between the input space and the output space (i.e. $f(\boldsymbol{I}) = \hat{\mathcal{Y}}$). Our main objective in machine learning is to predict the real mapping between the input space and the output space (i.e. $g(\boldsymbol{I}) = \mathcal{Y}$) so that Equation (2.3) holds true.

$$\forall(\boldsymbol{I}) \in \mathbf{R^W x R^H x R^N} : \qquad f(\boldsymbol{I}) = g(\boldsymbol{I}) \tag{2.3}$$

Machine learning algorithms use a loss function, $L(f(\ldots), g(\ldots))$,to penalize the deviation of the predicted output from the expected output. Let us consider Equation (2.1) as an example. If $(x_1, x_2) = (1, 2)$, then the expected output is $y = g(1, 2) = 3$. If the predicted output is $\hat{y} = f(1, 2) = 8$, then we could conclude that our prediction is wrong and a loss function could be used to quantize this error. Among the many possible alternatives, L_1 *loss* given in Equation (2.4) and L_2 *loss* given in Equation (2.5) could be used for our case. In the case of MxN dimensional output matrix $\boldsymbol{Y}$ and $\hat{\boldsymbol{Y}}$ where y_{ij} represents the element on ith row and jth column, Equation (2.6) and Equation (2.7) could be used for L_1 *loss* and L_2 *loss* respectively.

$$L_1(\hat{y}, y) = |\hat{y} - y| \tag{2.4}$$

$$L_2(\hat{y}, y) = (\hat{y} - y)^2 \tag{2.5}$$

$$L_1(\hat{y}, y) = \sum_{i=1}^{M} \sum_{j=1}^{N} |\hat{y_{ij}} - y_{ij}| \tag{2.6}$$

$$L_2(\hat{y}, y) = \sum_{i=1}^{M} \sum_{j=1}^{N} (\hat{y_{ij}} - y_{ij})^2 \tag{2.7}$$

Our objective is to find a mapping function which gives the minimum expected loss, $E(L(\hat{y}, y)|y)$ over the input space (conditioning on y is omitted in the following formulas for brevity). This expectation could be expressed with Equation (2.8) given that we want to find a mapping $f : x \to y$. The objective function could be formulated with Equation (2.9), where F is a set of possible functions which f is chosen from.

$$E(L(\hat{y}, y)) = \int_{x=-\infty}^{\infty} L(f(x), y)\, \mathrm{d}x \tag{2.8}$$

$$f^* = \underset{f \in F}{argmin} \quad E(\, L(f(x), y)\,) \tag{2.9}$$

Unfortunately, we do not have access to all data from the input space, so both Equation (2.8) and Equation (2.9) are intractable and we cannot find f^* [33].

Theorem 2.1. Weak Law of Large Numbers states that the expected value of independent and identically distributed (i.e. i.i.d.) random numbers $x_1, x_2, \ldots, x_n$ approaches to the mean of these numbers as $n \to \infty$ [34].

We could approximate the expected loss in Equation (2.8) with *Theorem* 2.1 given that input samples, $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, are $i.i.d$ and the number of samples, n, is large. The final formula is shown in Equation (2.10), where our dataset includes all input samples. Although we can find a function f^* where $L(f^*(x_i), y_i) = 0$ for $0 < i \leq n$, there is no guarantee that f^* will also minimize expected loss outside of our dataset. For example, we could define a function $\overline{g}(x)$ with Equation (2.11), where the expected outcome is 0 for even numbers and 1 for odd ones. If our dataset only contains even numbers, $\overline{f^*}(x) = 0$ will minimize the expected loss on the dataset ,but the minimization will not be generalizable to the odd numbers as well. This problem is described as *over-fitting*.

$$E(L(\hat{y}, y)) \approx \frac{1}{n} \sum_{i=1}^{n} L(f(x_i), y_i) \tag{2.10}$$

$$\overline{g}(x) = \begin{cases} 0 & \text{if } x \text{ is even,} \\ 1 & \text{if } x \text{ is odd.} \end{cases} \tag{2.11}$$

There are two main methods to prevent *over-fitting*. The first method suggests that we could split the dataset into three non-overlapping sets, namely *training*, *validation* and *test sets*. Instead of choosing the function which minimizes the expected loss over whole dataset, we could find a few different functions which minimize the loss on the *training set*. Then, the expected loss on the *validation set* could be utilized for choosing the best performing function. The intuition is that given a function *over-fits* on the *training set*, the loss on the *validation set* will be higher than the one on the *training set*. If the validation loss is similar to the training loss, the probability of *over-fitting* is small. The *test set* is used only once to determine the expected performance of that best performing function. The other method is called *regularization*, which includes any technique used to increase the probability of *generalization* even at the cost of increasing the training loss [35]. One particular technique is to add L_1 *norm*

or L_2 *norm* of the function parameters (i.e. θ) to Equation (2.9) for penalizing model complexity. We have presented final form of the objective function in Equation (2.12) where $R(f)$ is the regularization loss.

$$f^* = \underset{f \in F}{argmin} \quad \frac{1}{n} \sum_{i=1}^{n} L(f(x_i), y_i) + R(f) \tag{2.12}$$

Minimization of Equation (2.12) could be achieved with two methods for a given function $f(x)$. The first method utilizes a search space to determine the function parameters which gives the minimum loss. This method is not scalable, since the search space increase exponentially with the total number of parameters.

The other method is called gradient-based optimization [35]. For the sake of simplicity, suppose that we want to minimize the function $f(x) = y$ with respect to x instead of θ. We could determine whether the function $f(x)$ will increase when x is increased by a small positive number, ϵ, by examining the sign of $f'(x)$, namely its derivative. Let us examine Equation (2.13) to justify this statement. There are three cases:

(i) If $f(x)$ also increases so that $f(x + \epsilon) > f(x)$, $f'(x)$ becomes positive. If the function $f(x)$ stays relatively linear in range $[x - \epsilon, x + \epsilon]$, Equation 2.14 holds true. The inequality $f(x - \epsilon) < f(x)$ could be inferred from the same equation, since the left-hand side of it is positive. In this case, $f(x)$ decreases when x is decreased by ϵ.

(ii) If $f(x)$ stays the same so that $f(x + \epsilon) = f(x)$, $f'(x)$ becomes zero (this is called the *stationary point*). In this case, either increasing or decreasing x by ϵ does not change $f(x)$, if the relative linearity condition in *case* (1) holds.

(iii) If $f(x)$ decreases so that $f(x + \epsilon) < f(x)$, $f'(x)$ becomes negative. In this case, $f(x)$ decreases when x is increased by ϵ.

$$f'(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon} \tag{2.13}$$

$$f(x+\epsilon) - f(x) = f(x) - f(x-\epsilon) \tag{2.14}$$

Figure 2.2 shows $f(x) = \sin(x)$ and the sign of its derivative at x_1, $x_1 + \epsilon$, x_2, x_3 and x_4. Our objective is to minimize $f(x)$ by either increasing or decreasing x by ϵ. At point $x = x_1$, we could increase x_1 by ϵ to decrease $f(x_1)$ as suggested by *case* 3. At point $x = x_1 + \epsilon$, the sign of gradient did not change. We should again increase x by ϵ to decrease the function value. We could reach *the minimum of* $f(x)$, point x_2, by updating x in successive steps. At point $x = x_2$, the gradient is zero so that changing x by ϵ will not change $f(x)$ (given that ϵ is a very small number) as suggested by *case* 2. At point $x = x_3$,*case* 1 applies, so we could decrease x_3 by ϵ to decrease $f(x_3)$.*The minimum* of $f(x)$ is reached by updating x in successive steps. At point $x = x_4$,*case* 2 applies, but $f(x_4)$ decreases around $x = x_4$ as it can be seen in figure 2.2. The decline of $f(x)$ is not effective in range $[x - \epsilon, x + \epsilon]$, so we miss it. One way to work around this problem is to increase ϵ so that relative linearity does not hold true anymore.

One problem with this method is that, it is possible to stuck at *the local minimum* of $f(x)$ instead of finding *the global minimum*. *Figure* 2.3 shows that we could only reach *the local minimum*, x_2, by updating x_1 in successive steps by a small number. There are many heuristic methods to increase the probability of finding *the global minimum* as stated in [35].

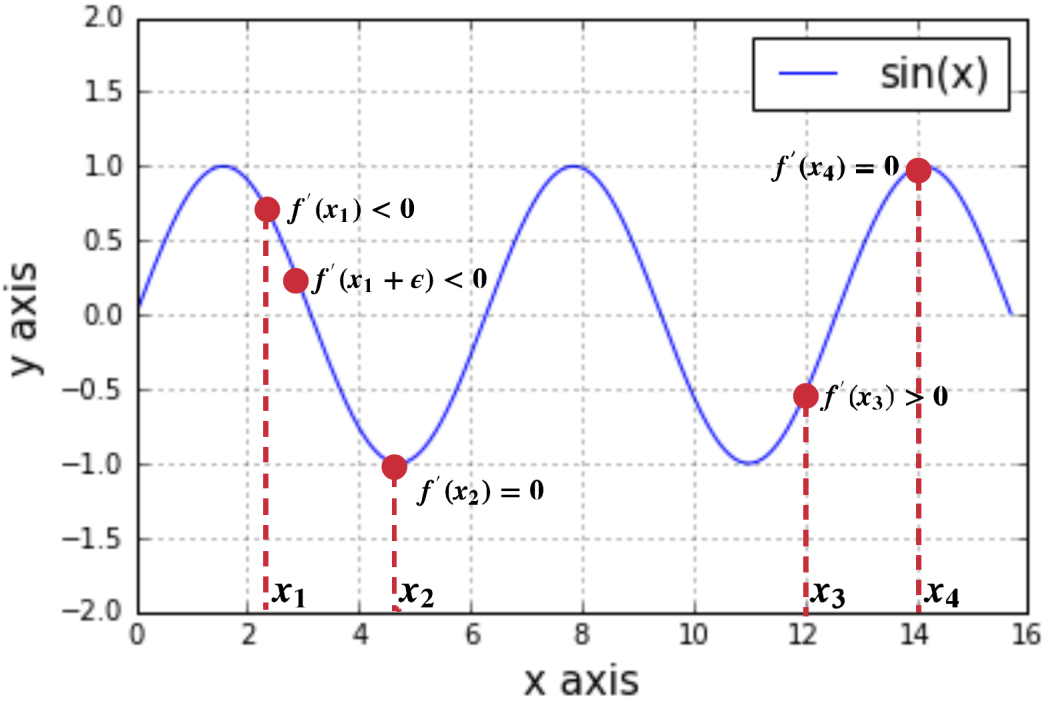

Figure 2.2. The plot of the function $y = \sin(x)$ for $x \in (0, 16)$

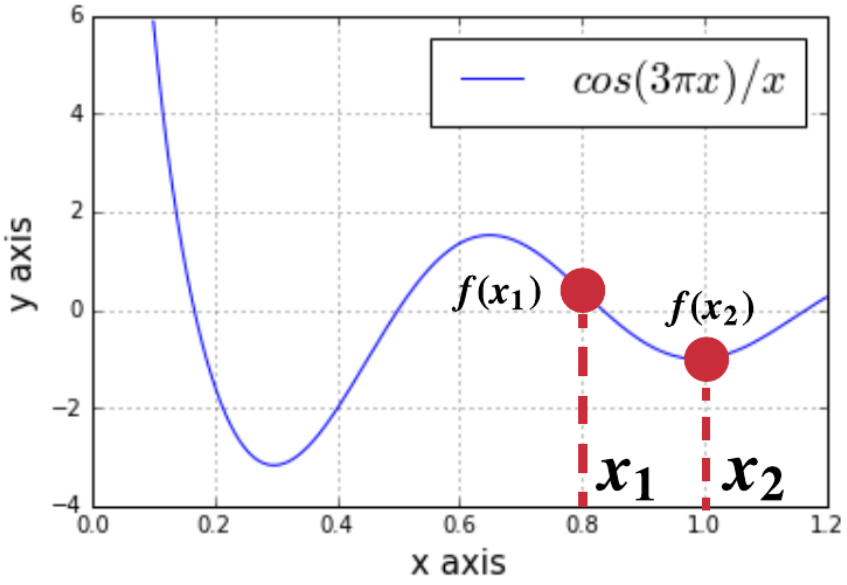

Figure 2.3. The plot of the function $y = cos(3\pi x)/x$ for $x \in [0, 1.2]$

Taylor series expansion [36] of function $f(x+\epsilon)$ about point x is presented in Equation (2.15). We want to minimize $f(x+\epsilon) - f(x)$ with respect to ϵ which is a very small number (not necessarily positive). In order to do that, the first order Taylor series expansion is used to determine ϵ, which produces the biggest decline in the value of $f(x)$. This method is called *gradient descent*.

$$f(x+\epsilon) = f(x) + f'(x)\,\epsilon + \frac{f''(x)}{2!}\,\epsilon^2 + \frac{f''(x)}{3!}\,\epsilon^3 + \ldots + \frac{f''(x)}{n!}\,\epsilon^n + \ldots \tag{2.15}$$

Proof. Let us use $K(x,\epsilon)$ to denote $f(x+\epsilon) - f(x)$. Now, from the Taylor series expansion formula, $K(x,\epsilon)$ could be expressed as

$$K(x,\epsilon) = f'(x)\,\epsilon + \frac{f''(x)}{2!}\,\epsilon^2 + \frac{f''(x)}{3!}\,\epsilon^3 + \ldots + \frac{f''(x)}{n!}\,\epsilon^n + \ldots \tag{2.16}$$

If the minimum of $K(x,\epsilon)$ with respect to ϵ exists, Equations (2.17) and (2.18) holds true.

$$\frac{d\,K(x,\epsilon)}{d\,\epsilon} = 0 \tag{2.17}$$

$$\frac{d^2\,K(x,\epsilon)}{d\,\epsilon^2}) > 0 \tag{2.18}$$

Let $O(n)$ represents Bachmann-Landau notation as stated by Erdélyi [37]. We can find the derivative of $K(x,\epsilon)$ as follows

$$\frac{d\,K(x,\epsilon)}{d\,\epsilon} = f'(x) + f''(x)\,\epsilon + \frac{f''(x)}{2!}\,\epsilon^2 + \ldots + \frac{f''(x)}{(n-1)!}\,\epsilon^{n-1} + \ldots \tag{2.19}$$

$$\frac{d\,K(x,\epsilon)}{d\,\epsilon} = f'(x) + f''(x)\,\epsilon + O(\epsilon^2) \tag{2.20}$$

If we use the *first order* Taylor series expansion, $O(\epsilon^2)$ could be ignored. The optimum value for ϵ could be derived by rewriting Equation (2.17).

$$f'(x) + f''(x)\,\epsilon = 0 \tag{2.21}$$

$$\epsilon = -\frac{f'(x)}{f''(x)} \tag{2.22}$$

Now, γ could be used instead of $\frac{1}{f''(x)}$ to reduce the computational complexity. Since we want to keep Equation (2.18) true, γ should be positive. Additionally, it should be small so that $f(x)$ stays relatively linear in range $[x-\epsilon,\ x+\epsilon]$. The optimum value for ϵ is given in Equation (2.23).

$$\epsilon = -\gamma\,f'(x) \tag{2.23}$$

□

Gradient descent states that $f(x+\epsilon)$ will be smaller than $f(x)$, given that Equation (2.23) is satisfied. In other words, we could reduce x by $\gamma\,f'(x)$ to minimize $f(x)$.

We have assumed that we want to minimize $f(x)$ with respect to x for simplicity. Our main objective is to minimize $f(x)$ with respect to function parameters, θ. Given that $f'(x)$ is the derivative of $f(x)$ with respect to θ, Equation (2.23) could be used to update θ in successive steps to reach the minimum of $f(x)$.

Let us summarize this section with the following points;

(i) Machine learning is used to find the latent mapping from an input space to an output space such that $f : x \rightarrow y$.
(ii) A loss function, $L(f(x), y)$, is used to quantize the deviation of the predicted output, $f(x)$, from the expected output, y.
(iii) Among the many techniques used to prevent *over-fitting*, regularization introduces an additional loss function, $R(f)$, to achieve generalization.
(iv) Our objective is to minimize the overall loss, as presented in Equation (2.12), on the training set. *Gradient descent* is one of the methods used for this minimization, which is achieved by reducing the loss gradually at successive time steps. The parameters of function $f(x)$ is updated at each step for this purpose.
(v) Expected loss on the validation set is used to prevent *over-fitting*.
(vi) Expected loss on the test set is used only once to determine the expected performance of the function on the unseen data.

2.2. Machine Learning Models

Machine learning algorithms could be split into two categories based on their objective. In the first category, namely *supervised learning*, we aim to learn the latent mapping $f : x \rightarrow y$ from an input space to an output space as demonstrated in the previous section. In this case, it is required that the dataset should contain each sample x_i with their corresponding *label* y_i. In the other case, these labels are also latent, so it is not possible to *supervise* the model with a loss function which depends on y. Therefore, the other category is called *unsupervised learning*.

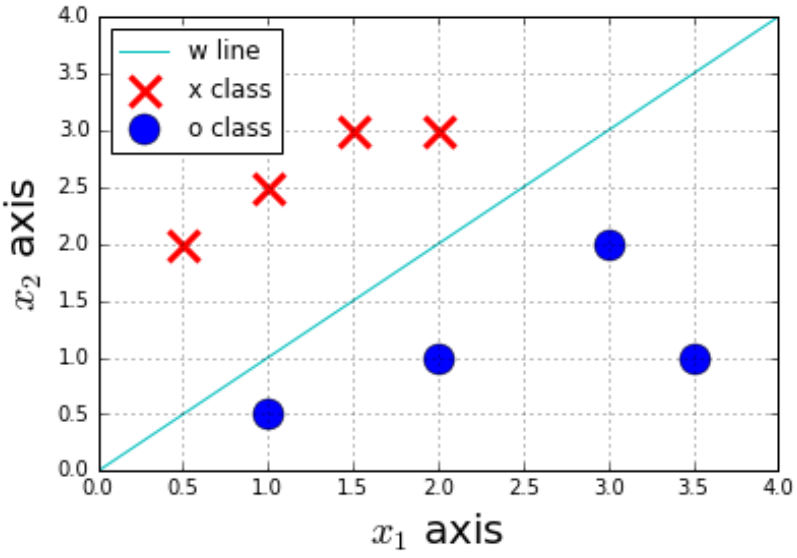

Figure 2.4. *Linear classifier example with two classes, "x" and "o"*

In this dissertation, supervised methods are used to build our description generation architecture. Specifically, *classification*, a subcategory of the supervised learning where the output space consists of finite alternatives for each label, is utilized. Because of that, only classification models are examined in this section. We advise Alpaydin [32] and Goodfellow *et al.* [35] to the readers for the explanation of other models.

2.2.1. Linear Classifier

Consider a simple example where our objective is to design a model to separate "x" and "o" symbols in Figure 2.4. In this case, the input space is two dimensional such that $\boldsymbol{x} \in \mathbf{R^2}$, where $\boldsymbol{x} = \{x_1, x_2\}$), and the output space is one dimensional such that $y \in \{x, o\}$. Since there are only two alternative outputs, we need to design a classification model for our problem.

We can separate one class from the another with line "w" as showed in Figure 2.4. In other words, "x" and "o" classes are *linearly separable*. Let us give the definition of *linear separability* [38] for the sake of completion.

Definition 1. Two classes $\mathcal{Y}_1$ and $\mathcal{Y}_2$ of R^d are *linearly separable* if there exists a hyperplane w of R^d such that the samples of $\mathcal{Y}_1$ and those of $\mathcal{Y}_2$ lie on the opposite side of it.

The separating line between two classes could be formulated with Equation (2.24), where w_1, w_2 and b are the parameters of this line. Let us divide the input space into three mutually exclusive and collectively exhaustive sets, namely $\mathscr{S}_1$, $\mathscr{S}_2$, $\mathscr{S}_3$, as expressed with Equation (2.25). Set $\mathscr{S}_2$ includes all the samples on this line, whereas set $\mathscr{S}_1$ and set $\mathscr{S}_3$ contain the samples from the opposite side of it. As suggested by the definition of *linear separability*, each sample in the dataset could be classified based on the sign of $l(\boldsymbol{x})$. Additionally, every sample in set $\mathscr{S}_2$ labeled as either "x" or "o" class in order to construct a classification rule which spans all the input space. Final classification rule is shown in Equation 2.26 (*Rule 1*).

$$l(\boldsymbol{x}) = w_1\, x_1 \,+\, w_2\, x_2 \,+\, b \,=\, 0 \tag{2.24}$$

$$l(\boldsymbol{x}) \begin{cases} < 0 & \text{if } \boldsymbol{x} \in \mathscr{S}_1, \\ = 0 & \text{if } \boldsymbol{x} \in \mathscr{S}_2, \\ > 0 & \text{if } \boldsymbol{x} \in \mathscr{S}_3. \end{cases} \tag{2.25}$$

$$y = \begin{cases} \text{“x”} & \text{if } \boldsymbol{x} \in \mathscr{S}_1, \\ \text{“o”} & \text{if } \boldsymbol{x} \in \{\mathscr{S}_2, \mathscr{S}_3\}. \end{cases} \tag{2.26}$$

Now that a classification rule is formulated, a loss function could be used to find the expected loss on the training set. We want to find the line "w^*" which minimizes a loss function, so that all samples in the dataset are classified correctly. Since there are infinitely many lines, which satisfies the linear separability definition, it is possible to pick one which overfits the training set. Intuitively, there is a higher probability of miss-classification for an unseen sample which is close to the line "w^*" than one which is far from it. Consider two points $\boldsymbol{x_a}$, $\boldsymbol{x_b}$ in the input space where $|l(\boldsymbol{x_a})| > |l(\boldsymbol{x_b})|$. Since the minimum distance between a point $\boldsymbol{x}$ and the separating line $l(\boldsymbol{x}) = 0$ is proportional to the absolute value of $l(\boldsymbol{x})$ as shown in Equation (2.27), we can say that the probability of making correct classification is higher for the point $\boldsymbol{x_a}$ then the point $\boldsymbol{x_b}$. Accordingly, we can use $l(\boldsymbol{x})$ as an input to a sigmoid function to get this probability which is given in Equation 2.28. The sigmoid function gives an output in the range [0,1] as shown in Figure 2.5.

$$d(\boldsymbol{x}) = \frac{|l(\boldsymbol{x})|}{\sqrt{w_1^2 + w_2^2}} \tag{2.27}$$

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{2.28}$$

We could interpret the output of sigmoid as the conditional probability of a sample being classified as either "x" or "o" given the sample coordinates (i.e. $p(y|\boldsymbol{x})$). Based on the final classification rule given in Equation (2.26), the conditional probability of "x" given the sample coordinates (i.e. $p(y = \text{"}x\text{"}|\boldsymbol{x})$) should be higher than the conditional probability of "o" given the sample coordinates (i.e. $p(y = \text{"}o\text{"}|\boldsymbol{x})$) for $\boldsymbol{x} \in \mathscr{S}_1$. Similarly the conditional probability of "o" given the sample coordinates should be higher than the conditional probability of "x" given the sample coordinates for $\boldsymbol{x} \in \{\mathscr{S}_2, \mathscr{S}_3\}$. Furthermore, the sum of conditional probabilities should be one (i.e.

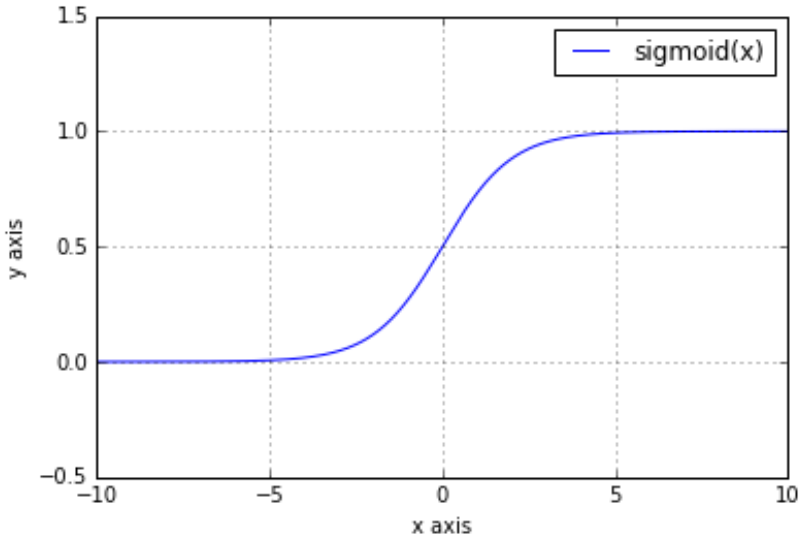

Figure 2.5. The plot of sigmoid function $\sigma(x) = \frac{1}{1+\exp(-x)}$

$p(y = \text{“}x\text{”}|\boldsymbol{x}) + p(y = \text{“}o\text{”}|\boldsymbol{x}) = 1)$. We could formulate the conditional probabilities based on the given conditions with Equations (2.29) and (2.30) (*Rule 2*).

$$p(y = \text{“}x\text{”}|\boldsymbol{x}) = \begin{cases} \sigma(|l(\boldsymbol{x})|) & \text{if } \boldsymbol{x} \in \mathscr{S}_1, \\ 1 - \sigma(|l(\boldsymbol{x})| & \text{if } \boldsymbol{x} \in \{\mathscr{S}_2, \mathscr{S}_3\}. \end{cases} \tag{2.29}$$

$$p(y = \text{“}o\text{”}|\boldsymbol{x}) = \begin{cases} 1 - \sigma(|l(\boldsymbol{x})|) & \text{if } \boldsymbol{x} \in \mathscr{S}_1, \\ \sigma(|l(\boldsymbol{x})| & \text{if } \boldsymbol{x} \in \{\mathscr{S}_2, \mathscr{S}_3\}. \end{cases} \tag{2.30}$$

Consider the classification rule in Equation (2.26) (*Rule 1*). In the case of $\boldsymbol{x} \in \mathscr{S}_1$, the label is always predicted as “x”, otherwise it is always predicted as “o”. In other words, the conditional probability of “x” given the sample coordinates is one (i.e. $p(y = \text{“}x\text{”}|\boldsymbol{x}) = 1$) for $\boldsymbol{x} \in \mathscr{S}_1$. Similarly, the conditional probability of “o” given the

sample coordinates is one, i.e. $p(y = \text{“}o\text{”}|\boldsymbol{x}) = 1$, for $\boldsymbol{x} \in \{\mathscr{S}_2, \mathscr{S}_3\}$. The conditional probabilities for *Rule 1* are given in Equations 2.31 and (2.32).

$$p(y = \text{“}x\text{”}|\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \in \mathscr{S}_1, \\ 0 & \text{if } \boldsymbol{x} \in \{\mathscr{S}_2, \mathscr{S}_3\}. \end{cases} \tag{2.31}$$

$$p(y = \text{“}o\text{”}|\boldsymbol{x}) = \begin{cases} 0 & \text{if } \boldsymbol{x} \in \mathscr{S}_1, \\ 1 & \text{if } \boldsymbol{x} \in \{\mathscr{S}_2, \mathscr{S}_3\}. \end{cases} \tag{2.32}$$

Comparing *Rule 1* with *Rule 2*, we see that the conditional probability of y given the sample coordinates is either zero or one (i.e. $p(y|\boldsymbol{x}) \in \{0, 1\}$) for *Rule 1*, whereas it can take the continuous values between zero and one (i.e. $p(y|\boldsymbol{x}) \in [0, 1]$) for *Rule 2*. Because of this reason, the former rule is also called *hard labeling*, whereas the latter one is called *soft labeling*. In both cases, our objective is to minimize the difference between the true conditional probability distribution $q(y|x)$, and predicted probability distribution $p(y|x)$. $q(y|x)$ could be considered as a deterministic hard labeling as shown in Equations (2.33) and (2.34).

$$q(y = \text{“}x\text{”}|\boldsymbol{x}) = \begin{cases} 1 & \text{if } y = \text{“}x\text{”}, \\ 0 & \text{if } y = \text{“}o\text{”}. \end{cases} \tag{2.33}$$

$$q(y = \text{“}o\text{”}|\boldsymbol{x}) = \begin{cases} 0 & \text{if } y = \text{“}x\text{”}, \\ 1 & \text{if } y = \text{“}o\text{”}. \end{cases} \tag{2.34}$$

We can use the *relative entropy* (i.e. *Kullback Leibler distance*) between these two distribution to quantize this difference as stated by Cover and Thomas [39]. The formula for the relative entropy is shown in Equation 2.35 and rewritten in Equation 2.36 as the difference of two terms. Since the first term on the right hand side of Equation 2.36 does not depend on the predicted probability $p(y|x)$, we cannot reduce its value with optimization. However, we could minimize the second term, namely *cross-entropy*. For these reasons, the optimization of the *relative entropy* is equivalent to the optimization of the *cross-entropy* and that is why instead of using the *relative entropy*, we build our loss function using the *cross-entropy* formula given in Equation (2.37). Since there are two possible classes for our problem, $H(q, p)$ is also named as *binary cross-entropy*.

$$D(q||p) = \sum_{y \in \{x,o\}} q(y|x) \log \frac{q(y|x)}{p(y|x)} \tag{2.35}$$

$$= \sum_{y \in \{x,o\}} q(y|x) \log q(y|x) - \sum_{y \in \{x,o\}} q(y|x) \log p(y|x) \tag{2.36}$$

$$H(q, p) = -\sum_{y \in \{x,o\}} q(y|x) \log p(y|x) \tag{2.37}$$

We could also simplify $H(q, p)$ by using Equations (2.33) and (2.34)

$$H(q,p) = \begin{cases} -\log q(y = \text{“}x\text{”}|\boldsymbol{x}) & \text{if } y = \text{“}x\text{”}, \\ -\log q(y = \text{“}o\text{”}|\boldsymbol{x}) & \text{if } y = \text{“}o\text{”}. \end{cases} \tag{2.38}$$

Now, we can construct the overall loss function with Equation (2.39). We could use either *Rule 1* or *Rule 2* for finding $p(y|x)$. The mapping $f^* : \boldsymbol{x} \rightarrow y$ which minimizes the loss function $L(\hat{y}, y)$ given in Equation (2.40) is found by applying the gradient descent on the loss function in successive steps.

$$L(\hat{y}, y) = \frac{1}{n} \sum_{i=1}^{n} H(q,p) + R(f) \tag{2.39}$$

$$f^* = \underset{f \in F}{argmin} \quad L(\hat{y}, y) \tag{2.40}$$

2.2.2. One Layer Neural Networks

We have seen that if the linear separability between the two classes exists, a linear classifier could be used to find the separating line between two classes. In case the linear separability does not exist, it is not possible to find such a line. Consider the classes “x” and “o” shown in Figure 2.6 as an example. These classes are obviously not linearly separable, and no single line could divide the input space such that the classification is achieved without error. However, we could use a circle with formula $x_1^2 + x_2^2 = r^2$ as a separating hyperplane to obtain zero classification error. In other words, we could use a non-linear hyperplane instead of linear one for classification in case that the classes are not linearly separable. Alternatively, we could use some

non-linear function to transform the input coordinates to another input coordinates where the linear separability between the two classes exists. For example, we have transformed the input coordinates $(x_1, x_2) \in \mathbf{R^2}$ in Figure 2.6 into the polar coordinates $(r, \theta) \in \{\mathbf{0}, \mathbf{R}^+\}\text{x}\{[-\pi, \pi)\}$ where $r = \sqrt{x_1^2 + x_2^2}$ and $\theta = \arctan \frac{x_2}{x_1}$ and show the result in Figure 2.7. After the transformation, a linear classifier could be used to find the separating line w, similar to the previous section. Now, the problem is reduced to designing a non-linear transformation followed by a linear classifier as shown in Figure 2.8.

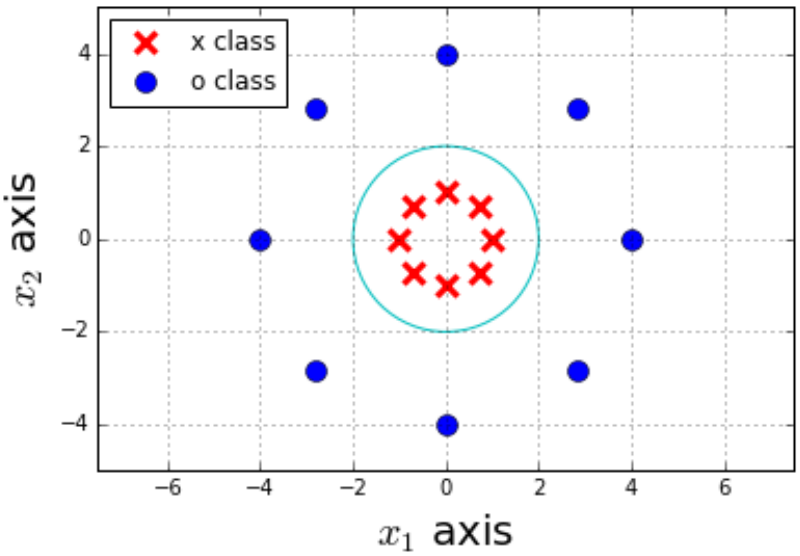

Figure 2.6. The plot of samples belonging either "x" or "o" classes which are not linearly separable for the input coordinates $(x_1, x_2) \in \mathbf{R^2}$

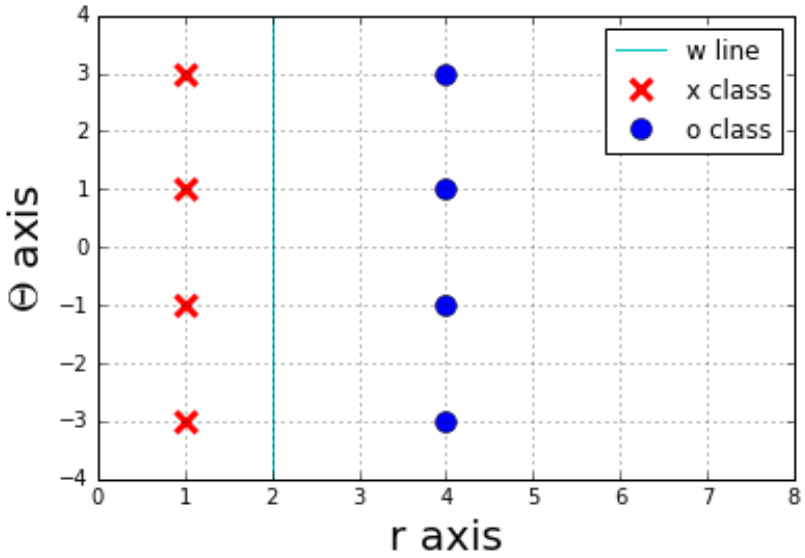

Figure 2.7. The plot of samples belonging either "x" or "o" classes which are linearly separable for polar input coordinates $(r, \theta) \in \{\mathbf{0}, \mathbf{R}^+\}\mathrm{x}\{[-\pi, \pi)\}$ where $r = \sqrt{x_1^2 + x_2^2}$ and $\theta = \arctan \frac{x_2}{x_1}$

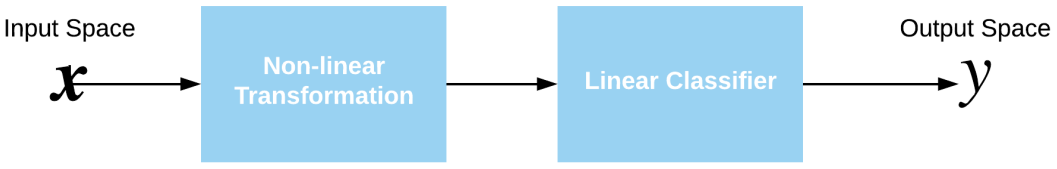

Figure 2.8. Non-linear classification pipeline

The architecture of an one hidden layer neural network is shown in Figure 2.9. Our objective is to find a non-linear mapping from an input space to an output space in order to separate two classes. Each circle in the architecture is named as *neuron* or *node* and each line between the neurons is named as *weight*. Apart from the first layer, each neuron takes an input and gives an output based on some *activation function*. Each neuron is also named after their output for convenience (e.g. the output of the neuron on the upper left corner is x_1).

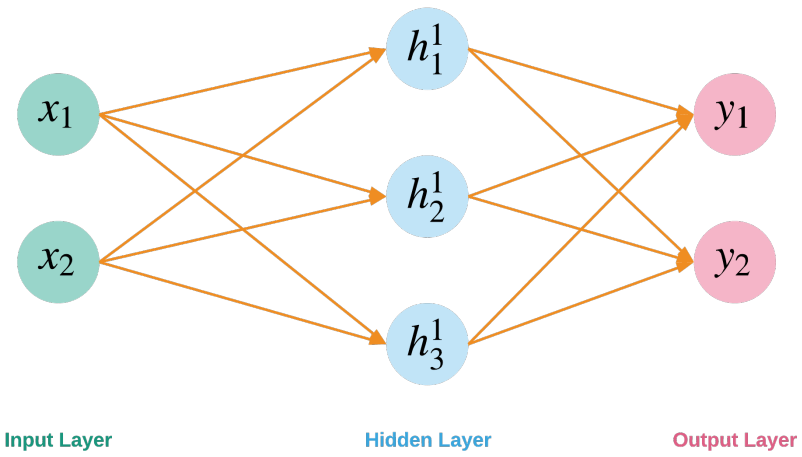

Figure 2.9. One layer neural network architecture

The number of neurons in the first layer, namely *input layer*, equals to the number of input features $\boldsymbol{x} = [x_1, x_2]$. The output of each neuron equals to the value of one feature. The number of neurons in the second layer, namely *hidden layer*, depends on the design of the architecture. Since each neuron in this hidden layer is connected to all the neurons in the previous layer, this layer is also called a *fully connected* (FC) layer. The non-linear transformation of the classification pipeline shown in Figure 2.8 is performed in this layer. Each neuron on the hidden layer takes an input s_i^k, passes it through some non-linear function $f^k()$ and gives the result h_i^k as the output such that $h_i^k = f^k(s_i^k)$. The subscript i and superscript k specifies that $h_i^k, f^k(), s_i^k$ are the parameters of i^{th} neuron belonging to k^{th} layer after the input layer. Input layer is declared as 0^{th} layer. The number of neurons in the last layer, namely *output layer*, equals to the number of classes that needs to be classified. For example, there should be

four neurons at the output layer given that we want to separate 4 different classes. The linear classification shown in Figure 2.8 is performed in this layer. Depending on the output activation function, y_i could be interpreted as either the confidences score of a sample belonging to i^{th} class, represented with a $y_i \in \mathbf{R}$, or the conditional probability of a sample belonging to i^{th} class given the input coordinates $\boldsymbol{x}$, represented with a $y_i \in [0, 1]$. The later one is generally obtained with the normalization of the confidence scores such that after the normalization, $\sum_{i=1}^{N} y_i = 1$ for N different classes.

The input of the neuron h_j^1 on the hidden layer can be expressed as the summation $s_j^1 = \sum_{i=1}^{2} w_{ij}^1 x_i + b_j^1$, where w_{ij}^1 is the weight between i^{th} input neuron x_i and j^{th} hidden neuron h_j^1, and b_j^1 is the *bias* of h_j^1. This summation could be rewritten as a vector multiplication $s_j^1 = \boldsymbol{w_j^1}^T \boldsymbol{x}$, where $\boldsymbol{x} = [x_1, x_2, 1]^T$ and $\boldsymbol{w_j^1} = [w_{1j}^1, w_{2j}^1, b_j^1]^T$. In order to further simplify our representation, Equations (2.41), (2.42) and (2.43) could be combined into a matrix-vector multiplication $\boldsymbol{s^1} = \boldsymbol{W^1 x}$ as shown in Equation (2.44), where $\boldsymbol{s^1} = [s_1^1, s_2^1, s_3^1]^T$ and $\boldsymbol{W^1} = [\boldsymbol{w_1^1}, \boldsymbol{w_2^1}, \boldsymbol{w_3^1}]^T$.

$$s_1^1 = \boldsymbol{w_1^1}^T \boldsymbol{x} \tag{2.41}$$

$$s_2^1 = \boldsymbol{w_2^1}^T \boldsymbol{x} \tag{2.42}$$

$$s_3^1 = \boldsymbol{w_3^1}^T \boldsymbol{x} \tag{2.43}$$

$$\underbrace{\begin{bmatrix} s_1^1 \\ s_2^1 \\ s_3^1 \end{bmatrix}}_{\boldsymbol{s^1}} = \underbrace{\begin{bmatrix} w_{11}^1 & w_{21}^1 & b_1^1 \\ w_{12}^1 & w_{22}^1 & b_2^1 \\ w_{13}^1 & w_{23}^1 & b_3^1 \end{bmatrix}}_{\boldsymbol{W^1}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}}_{\boldsymbol{x}} \tag{2.44}$$

The non-linear function $f(\ldots)$ is usually chosen as a convex function since the local minimum and the global minimum of a convex function are the same, so it is easier to optimize the loss function. For a comprehensive introduction to convex optimization, we recommend Convex Optimization book from Boyd and Vandenberghe [40]. We have listed some popular activation functions used in neural networks below.

(i) sigmoid

Sigmoid function given in Equation (2.45) is defined from the input space $x \in \mathbf{R}$ to the output space $f(x) \in [0, 1]$ as shown in Figure 2.10.

There are two main problems with the *sigmoid* function when it is used as an activation function. It is shown by LeCun *et al.* [41] that the output of an activation function should be zero centered in order to reduce the number of steps required for optimization. The *sigmoid* function is not zero centered. Another problem rises from the fact that the gradient of the *sigmoid* function for the majority of the input space is almost zero, which in turn makes the gradient of the loss function very small. Although *sigmoid* function was widely used as an activation function over the past few years [35], it is not preferred anymore for these reasons.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.45}$$

(ii) tanh

LeCun *et al.* [41] used *tanh* as an activation function in order to eliminate the problems arising from the fact that the output of the *sigmoid* function is not zero-centered. tanh function given in Equation (2.46) is defined from the input space $x \in \mathbf{R}$ to the output space $f(x) \in [-1, 1]$ as shown in Figure 2.11. Although the output of this function is zero centered, the gradient of it is also almost zero for the majority of the input space.

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.46}$$

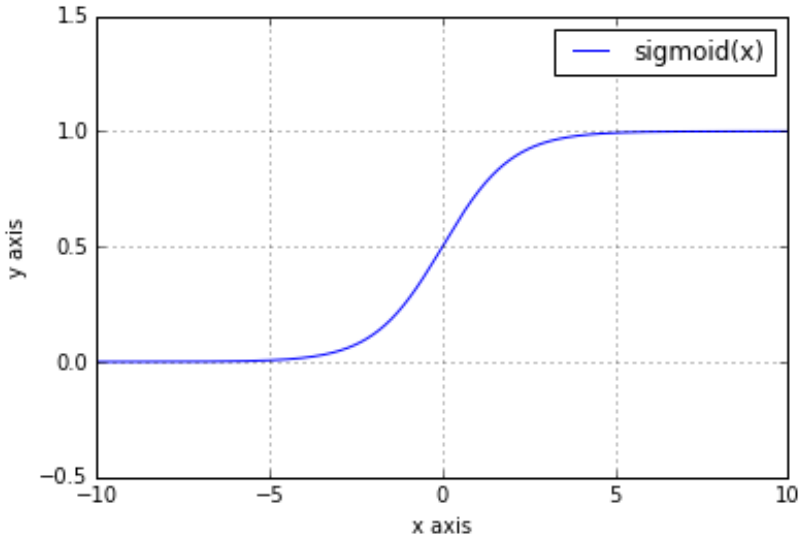

Figure 2.10. The plot of sigmoid function $\sigma(x) = \frac{1}{1+\exp(-x)}$

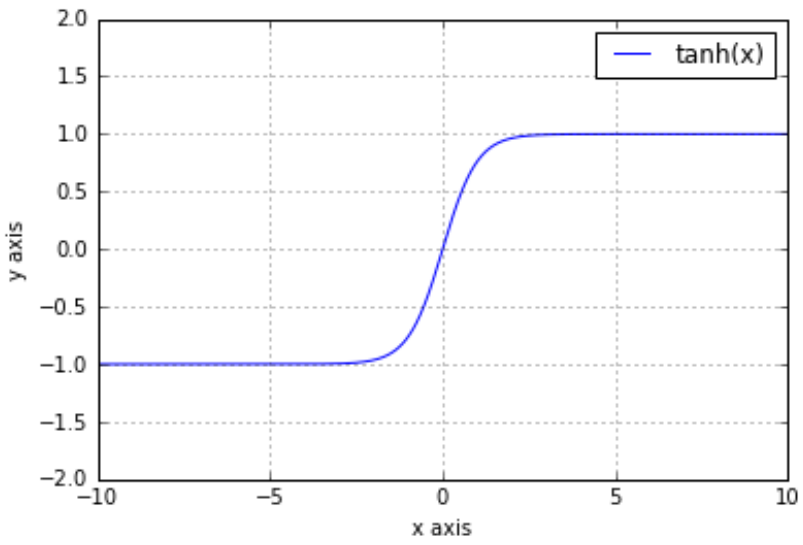

Figure 2.11. The plot of tanh function $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

(iii) ReLU (Rectified Linear Unit)

ReLU function given in Equation (2.47) is defined from the input space $x \in \mathbf{R}$ to the output space $f(x) \in [0, +\infty]$ as shown in Figure 2.12. Nair and Hinton [42] used ReLU activations to increase the performance of the restricted boltzmann machines, whereas Krizhevsky *et al.* [5] achieved the best performance on both the classification and the localization tasks on ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC-2012) [4].

Although the output of the ReLU is not zero-centered, the gradient is not zero for the half of the input space (i.e. $x \in \mathbf{R}^+$) which in turn reduces the number of iterations required to reach the minimum of loss function.

$$f(x) = \max(0, x) \tag{2.47}$$

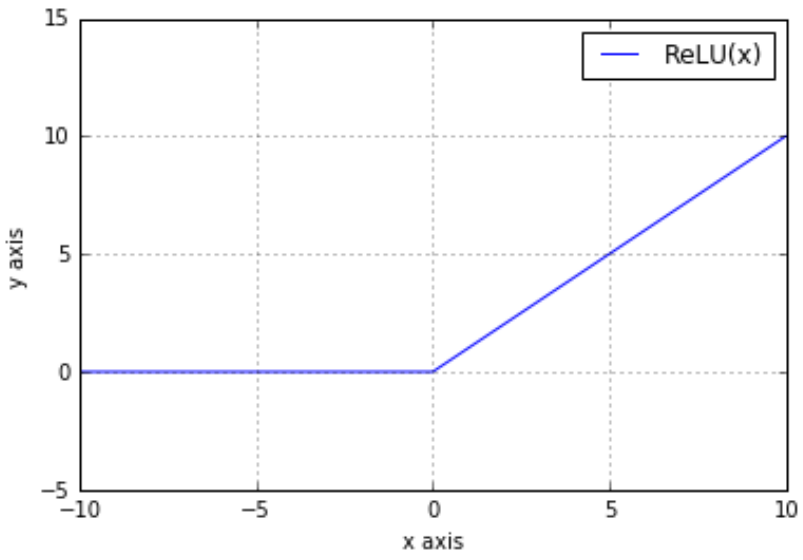

Figure 2.12. The plot of ReLU function $f(x) = \max(0, x)$

(iv) LReLU (Leaky ReLU)

Maas *et al.* [43] verified that LReLU given in Equation (2.48) converges slightly faster than the ReLU resulting from the non-zero gradients for $x \in \mathbf{R}^-$ as shown in Figure 2.13.

$$f(x) = \begin{cases} 0.1x & \text{if } x < 0, \\ x & \text{if } x \geq 0 \end{cases} \tag{2.48}$$

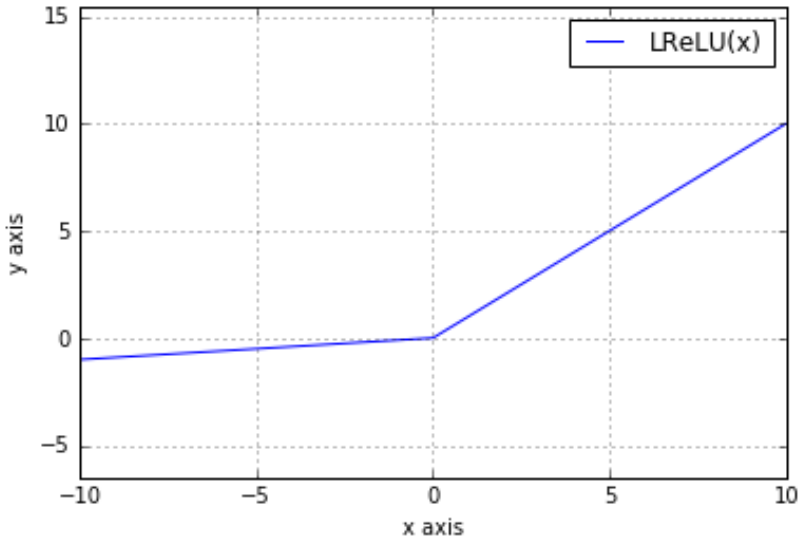

Figure 2.13. The plot of LReLU function

(v) PReLU (Parametric ReLU)

As suggested by He *et al.* [44], PReLU function given in Equation (2.49) uses α as a parameter of the activation function in order to avoid zero gradients. α is optimized with the gradient descent.

$$f(x) = \begin{cases} \alpha x & \text{if } x < 0, \\ x & \text{if } x \geq 0 \end{cases} \tag{2.49}$$

(vi) ELU (Exponential Linear Units)

Clevert *et al.* [45] showed that ELU activation function given in Equation (2.50) ensures robustness to noise unlike ReLU, LReLU and PReLU without sacrificing the generalization performance. α is a constant positive number.

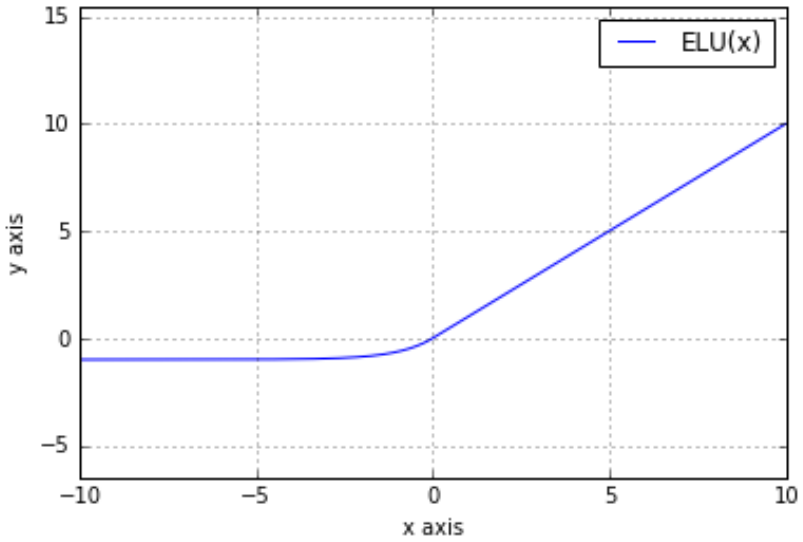

Figure 2.14. The plot of ELU function

$$f(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x < 0, \\ x & \text{if } x \geq 0 \end{cases} \tag{2.50}$$

Although it is not necessary, usually the same activation function is used for each neuron in the same layer. The hidden layer outputs are calculated using Equations (2.51), (2.52) and (2.53). These equations could be rewritten as $\boldsymbol{h^1} = f^1(\boldsymbol{W_1 x})$, where $\boldsymbol{h^1} = [h_1^1, h_2^1, h_3^1]^T$ and f^1 is applied element-wise to the vector $\boldsymbol{W^1 x}$. The mathematical formulation between the input layer and the hidden layer is summarized in Figure 2.15

$$h_1^1 = f^1(\boldsymbol{w_1^1}^T \boldsymbol{x}) \tag{2.51}$$

$$h_2^1 = f^1(\boldsymbol{w_2^1}^T \boldsymbol{x}) \tag{2.52}$$

$$h_3^1 = f^1(\boldsymbol{w_3^1}^T \boldsymbol{x}) \tag{2.53}$$

The input of the neuron y_j on the output layer can be expressed as the summation $s_j^2 = \sum_{i=1}^{3} w_{ij}^2 h_i^1 + b_j^2$, where w_{ij}^2 is the weight between i^{th} hidden neuron h_i^1 and j^{th} output neuron y_j, and b_j^2 is the *bias* of y_j. This summation can also be expressed as a matrix-vector multiplication $\boldsymbol{s^2} = \boldsymbol{W^2 h^1}$ with Equation (2.54).

$$\underbrace{\begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix}}_{\boldsymbol{s_2}} = \underbrace{\begin{bmatrix} w_{11} & w_{21} & w_{31} & b_1 \\ w_{12} & w_{22} & w_{31} & b_2 \\ w_{13} & w_{23} & w_{31} & b_3 \end{bmatrix}}_{\boldsymbol{W_2}} \underbrace{\begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ 1 \end{bmatrix}}_{\boldsymbol{h_1}} \tag{2.54}$$

As mentioned before, depending on the activation function $f^2(\ldots)$, y_i could be interpreted as either the confidence score of a sample belonging to i^{th} class or the conditional probability of a sample belonging to i^{th} class given the input coordinates $\boldsymbol{x}$. For the first case, we could use the identity mapping $f(x) = x$ as the activation function. On an intuitive level, each neuron in the output layer splits the input space with a hyperplane and the confidence score represents the distance between a sample and this hyperplane. *Multiclass SVM loss* [46] shown in Equation (2.55) could be used

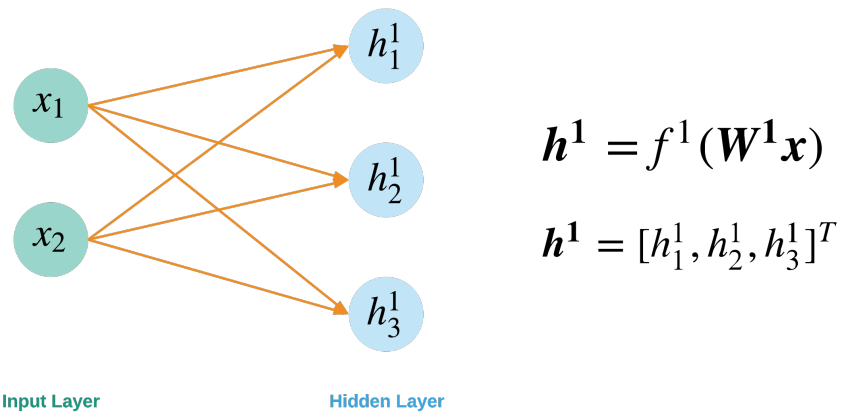

Figure 2.15. Mathematical formulation between the input layer and the hidden layer

to calculate the loss of a particular sample, where N is the number of output neurons (e.g. 2 for our classification problem), $k \in \{1, 2, \ldots, N\}$ is the true class of that sample and Δ is a constant scalar, namely *margin*. Notice that the loss becomes zero when the confidence score of the true class y_k gets higher than the confidence score of all the other classes by margin Δ. A sample is classified with the class i, if the confidence score of i^{th} class (i.e. y_i) is higher than the confidence score of all the other classes.

$$L = \sum_{\substack{i=1 \\ i \neq k}}^{N} max(0, y_i - y_k + \Delta) \tag{2.55}$$

For the second case, the softmax function [35] given in Equation (2.56) could be used as the activation function in order to obtain the conditional probabilities. N in this equation equals to the number of output neurons. Similar to the linear classifier, we could use the cross-entropy loss for the optimization. Equation (2.57) shows the multiclass cross entropy loss, where y is the *true* class label, $\hat{y}$ is the *predicted* class label, $q(y = i|\boldsymbol{x})$ is the *true* conditional probability of a sample belonging to i^{th} class and $p(\hat{y} = i|\boldsymbol{x})$ is the *predicted* conditional probability of a sample belonging to i^{th} class such that $p(\hat{y} = i|\boldsymbol{x}) = y_i$. Assuming that $k \in \{1, 2, \ldots, N\}$ is the true class of a sample, $q(y = i|\boldsymbol{x})$ becomes deterministic so that $q(y = k|\boldsymbol{x}) = 1$ and $q(y = i|\boldsymbol{x}) = 0$ for $i \neq k$. We could simplify Equation (2.57) to Equation (2.58) by explicitly writing each term of the summation. A sample is classified with the class i, if the conditional probability of i^{th} class (i.e. y_i) is higher than the conditional probability of all the other classes.

$$f(s_j) = \frac{e^{s_j}}{\sum_{i=1}^{N} e^{s_i}} \tag{2.56}$$

$$L = -\sum_{i=1}^{N} q(y = i|\boldsymbol{x}) \log p(\hat{y} = i|\boldsymbol{x}) \tag{2.57}$$

$$L = -\log(\frac{e^{s_k}}{\sum_{i=1}^{N} e^{s_i}}) = -\log y_i \tag{2.58}$$

The mathematical formulation between the hidden layer and the output layer is summarized in *Figure* 2.16 for convenience.

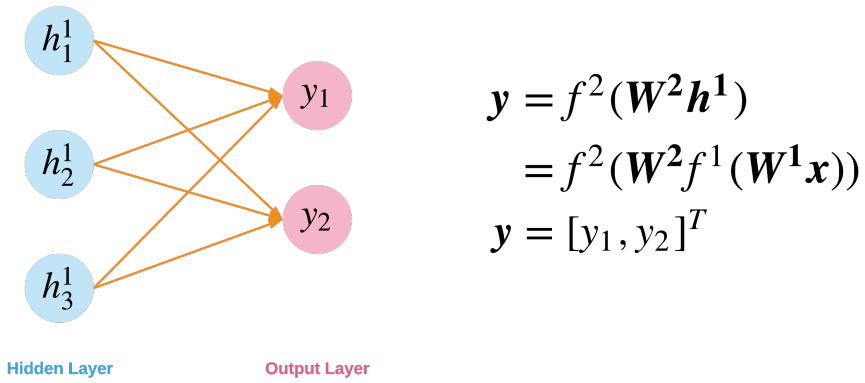

Figure 2.16. Mathematical formulation between the hidden layer and the output layer

2.2.3. Multi-Layer Neural Networks

In some cases, the accuracy achieved with one layer neural network could be non-sufficient for our classification problem. Although it is not always true, we could get a better classification accuracy when the number of hidden layers increases. In particular, He *et al.* [9] showed that as the number of hidden layers increases, the accuracy could decrease because it gets harder to optimize the parameters of a neural network. Although we will not dive into the details, they have also showed a way to overcome this problem in their work [9]. For our case, we could use two layer neural network shown in Figure 2.17 to increase the classification accuracy. For simplicity, the same network is also shown as a block diagram in Figure 2.18, where ReLU activation function is used in the hidden layers and softmax activation function is used in the

output layer. We have used "FC-M" to indicate the fully connected structure of a layer whose output is a "M" dimensional vector. For instance, FC-2 performs the operation $\boldsymbol{s^3} = \boldsymbol{W^3 h^2}$ between the hidden layer 2 and output layer, where $\boldsymbol{s^3}$ is a two dimensional vector.

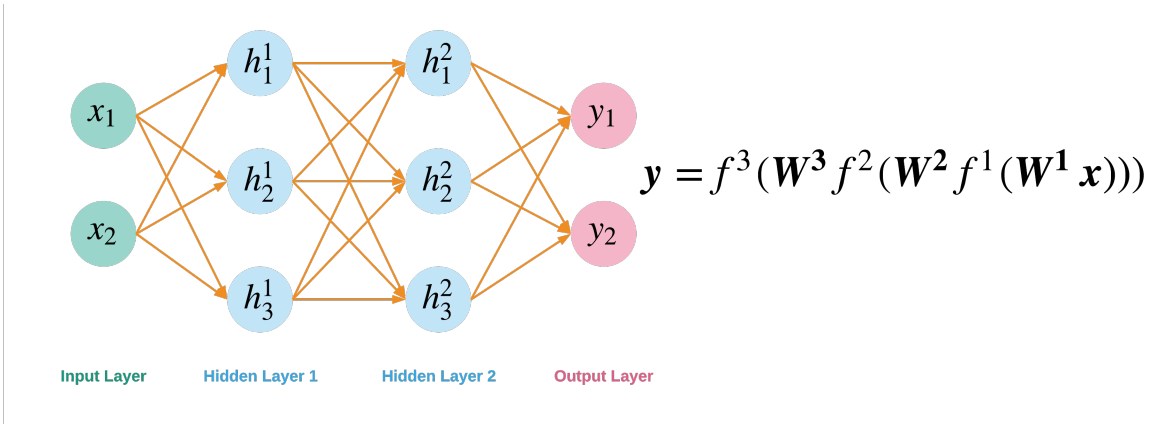

Figure 2.17. A neural network architecture with two layers

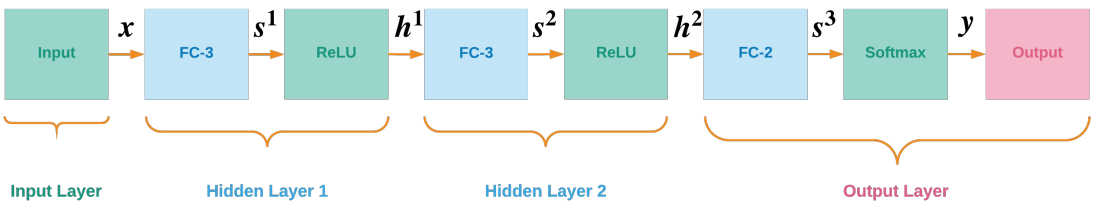

Figure 2.18. Another way to represent a neural network architecture with two layers

2.2.4. Convolutional Neural Networks

As stated earlier, each neuron in a FC layer is connected to all neurons in the previous layer. In other words, each neuron in a FC layer *sees* all the neurons in the previous layer. Formally stated, FC layers have *global receptive field*. For example, a neuron in a FC layer, which comes after WxHxN dimensional input image, has WxHxN dimensional receptive field. Consequently, FC layers do not preserve the spatial structure of the input, such as images and sound spectrogram.

Lecun *et al.* [6] showed that convolutional neural networks (CNNs) could be used to preserve spatial structure of the input by ensuring shift, scale and distortion invariance to some degree. Now, we will explain two main layers used in CNNs, namely convolutional and pooling layers.

(i) Convolutional (CONV) layer performs the convolution operation on its input with a set of filters. First, we will show this operation with an example by convolving a WxHxN dimensional image $\boldsymbol{X}$ with a F_1xF_2xN dimensional filter $\boldsymbol{W}$. Then, we will generalize this result to a case where a WxHxN dimensional input $\boldsymbol{X}$ is convolved with F_1xF_2xNxK dimensional filters $\boldsymbol{W}$, where the last dimension represents the total number of filters.

Consider 8x8x3 dimensional cat image $\boldsymbol{X}$ shown in Figure 2.19, where each channel of the image is represented with $\boldsymbol{X_k}$ such that$\boldsymbol{X_k} = \boldsymbol{X}[:, :, k]$. We will convolve this image with 3x3x3 dimensional filter $\boldsymbol{W}$ shown in Figure 2.20, where each channel of the filter is represented with $\boldsymbol{W_k}$ such that $\boldsymbol{W_k} = \boldsymbol{W}[:, :, k]$. We have omitted the bias vector for brevity in this example.

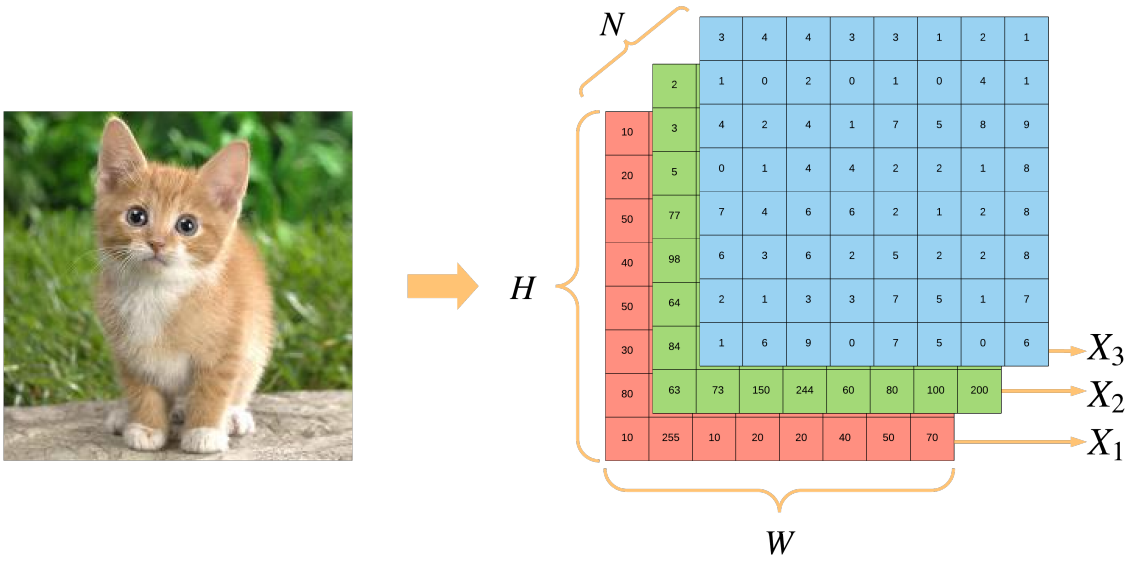

Figure 2.19. A cat image which is represented with a 8x8x3 dimensional matrix $\boldsymbol{X}$

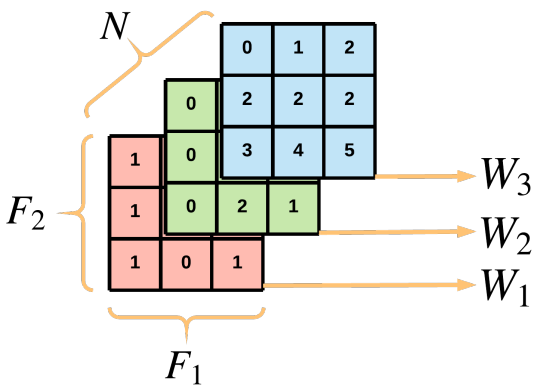

Figure 2.20. A convolutional filter which is represented with 3x3x3 dimensional matrix $\boldsymbol{W}$

Figure 2.21 illustrates convolution operation for the calculation of $\boldsymbol{S}[1,1]$, where $\boldsymbol{S}$ is the output of *CONV operation* (not to be confused with $\boldsymbol{H}$, i.e. the output of *CONV layer*, as shown in Figure 2.22). Initially, each filter channel $\boldsymbol{W_k}$ is element-wise multiplied with a 3x3 portion of corresponding input channel $\boldsymbol{X_k}$ (highlighted 3x3 matrices in Figure 2.21). The resulting 3x3 matrix is represented with $\boldsymbol{L_{k-ij}}$, where $\boldsymbol{ij}$ indicates that this operation is performed for the calculation of $\boldsymbol{S[i,j]}$. Each element of $\boldsymbol{L_{k-11}}$ in Figure 2.21 is calculated with Equation (2.59). Since there are 3 channels in this example, a total of 3x3x3 element-wise multiplication is performed. In other words, each neuron in this layer *sees* only 3x3x3 neurons in the previous layer. Formally stated, CONV layers have *local receptive field* (e.g. 3x3x3 in this example). Additionally, $\boldsymbol{X}$ and $\boldsymbol{W}$ should have the same number of channels in order to perform the convolution operation. Afterwards, the elements of $\boldsymbol{L_{1-11}}$, $\boldsymbol{L_{2-11}}$ and $\boldsymbol{L_{3-11}}$ are added together to calculate $\boldsymbol{S}[1,1]$ as shown in Equation (2.60).

$$\boldsymbol{L_{k-11}}[m,n] = \boldsymbol{X_k}[m,n] \cdot \boldsymbol{W_k}[m,n] \qquad \text{for } m,n \in [1,3] \tag{2.59}$$

$$\boldsymbol{S}[1,1] = \sum_{k=1}^{3}\sum_{m=1}^{3}\sum_{n=1}^{3} \boldsymbol{L_{k-11}}[m,n] \tag{2.60}$$

Figure 2.23 illustrates convolution operation for the calculation of $\boldsymbol{S}[1,2]$. Similar to the previous calculation, each filter channel $\boldsymbol{W_k}$ is element-wise multiplied with a 3x3 portion of the input channel $\boldsymbol{X_k}$ (highlighted 3x3 matrices in Figure 2.23). Unlike to the previous calculation, the 3x3 portion of $\boldsymbol{X_k}$ is now shifted one pixel to the right as shown in Equation (2.61). In other words, we slide the filter $\boldsymbol{W_k}$ by one element to the right on $\boldsymbol{X_k}$ in order to calculate one element shifted version of $\boldsymbol{S}[1,1]$, namely $\boldsymbol{S}[1,2]$. The number of elements which the filter is slided horizontally comparing the computation of $\boldsymbol{S}[i,j]$ and $\boldsymbol{S}[i,j+1]$ is called *horizontal stride*. Similarly, the number of elements which the filter is slided vertically comparing the computation of $\boldsymbol{S}[i,j]$ and $\boldsymbol{S}[i+1,j]$ is called *vertical stride*. In this example, the convolutional filter has horizontal and vertical stride one. $\boldsymbol{S}[1,2]$ is also calculated similar to the calculation of $\boldsymbol{S}[1,1]$ (i.e. the elements

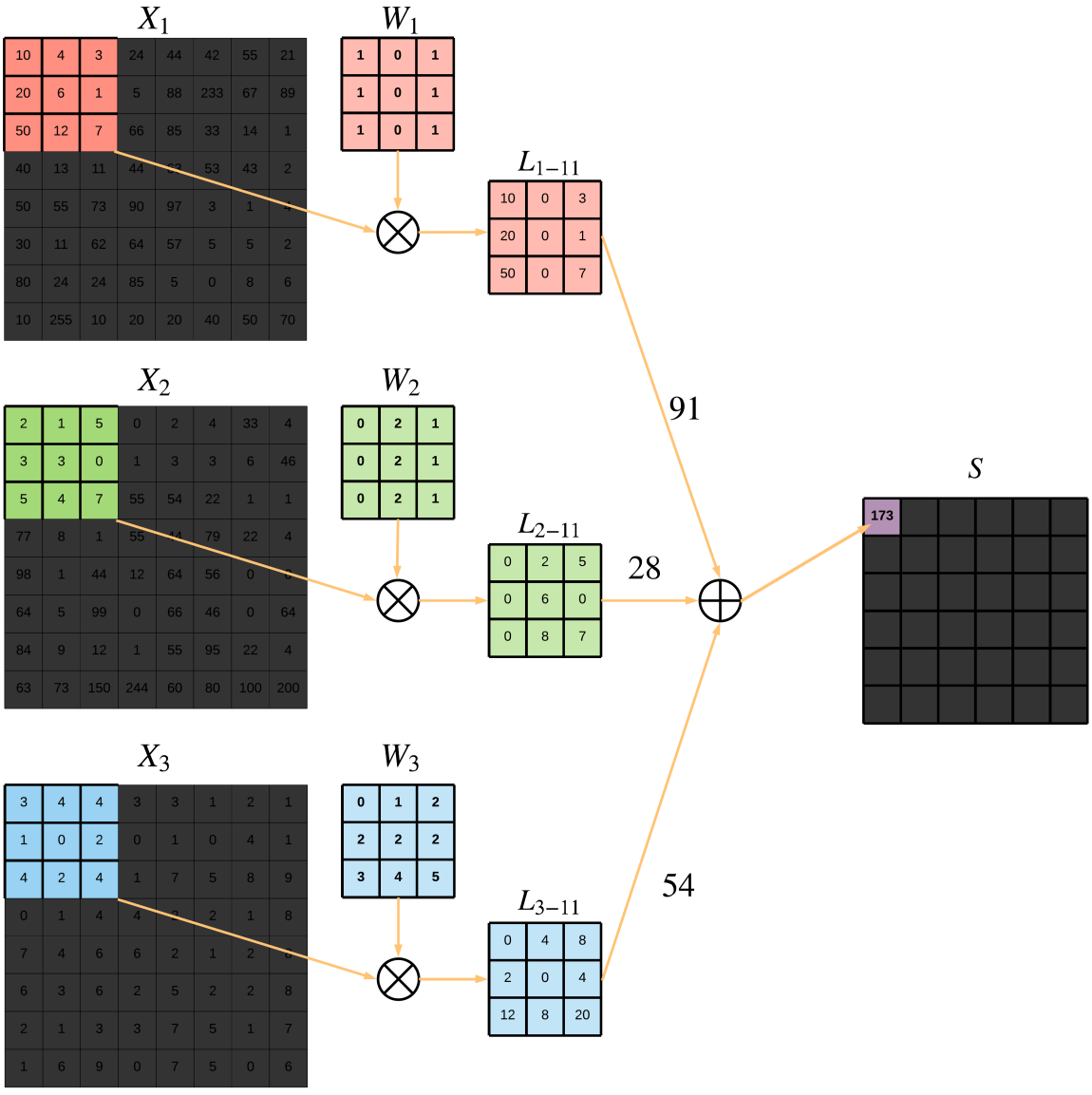

Figure 2.21. Illustration of the convolution operation for the calculation of $\boldsymbol{S}[1,1]$

of $\boldsymbol{L_{1-12}}$, $\boldsymbol{L_{2-12}}$ and $\boldsymbol{L_{3-12}}$ are added together as shown in Equation (2.62)).

$$\boldsymbol{L_{k-12}}[m,n] = \boldsymbol{X_k}[m,n+1] \cdot \boldsymbol{W_k}[m,n] \qquad \text{for } m,n \in [1,3] \tag{2.61}$$

$$\boldsymbol{S}[1,2] = \sum_{k=1}^{3}\sum_{m=1}^{3}\sum_{n=1}^{3} \boldsymbol{L_{k-12}}[m,n] \tag{2.62}$$

We could compute every element of $\boldsymbol{S}$ by sliding 3x3 filter $\boldsymbol{W_k}$ on 8x8 input $\boldsymbol{X_k}$. Considering that we are using horizontal and vertical stride one for the convolution, $\boldsymbol{W_k}$ could be slided 6 times horizontally and 6 times vertically. This will generate a 6x6 dimensional matrix $\boldsymbol{S}$, whose elements are calculated with Equations (2.63) and (2.64), where $k,m,n \in [1,3]$ and $i,j \in [1,6]$. Finally, *ReLU* activation function $f(\ldots)$ is applied element-wise to $\boldsymbol{S}$ to calculate the output of *CONV layer* ,i.e. $\boldsymbol{H}$, as shown in Equation (2.65).

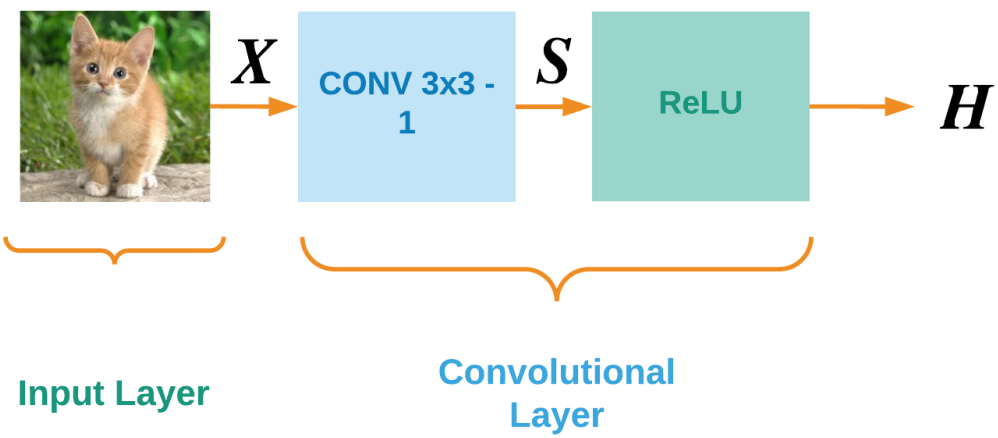

Figure 2.22. The output of convolutional layer in this example, $\boldsymbol{H}$, is calculated after the input image, $\boldsymbol{X}$, is passed through a CONV operation followed by a ReLU activation function

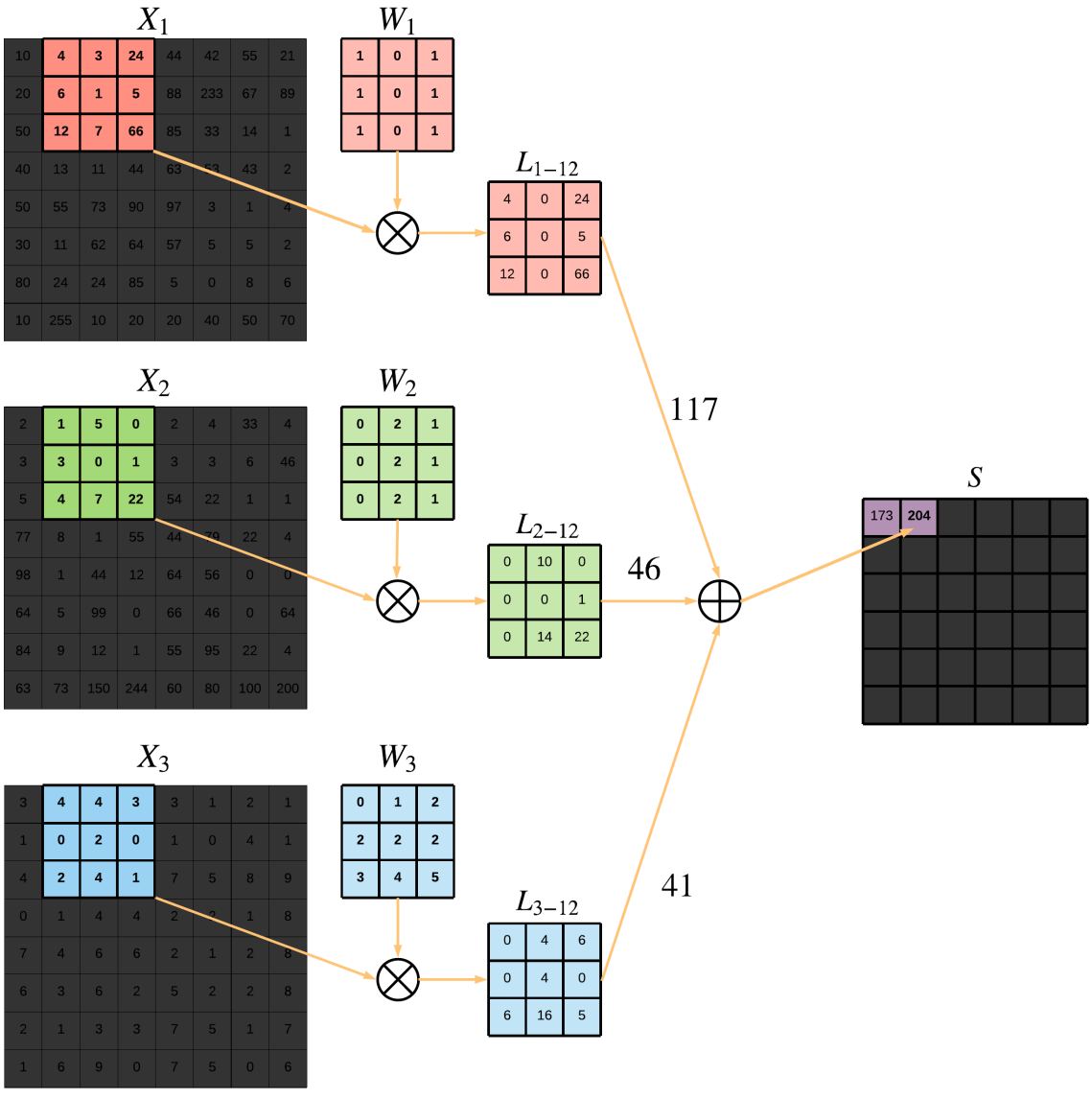

Figure 2.23. Illustration of the convolution operation for the calculation of $\boldsymbol{S}[1, 2]$

$$\boldsymbol{L_{k-ij}}[m,n] = \boldsymbol{X_k}[m+i-1, n+j-1] \cdot \boldsymbol{W_k}[m,n] \tag{2.63}$$

$$\boldsymbol{S}[i,j] = \sum_{k=1}^{3}\sum_{m=1}^{3}\sum_{n=1}^{3} \boldsymbol{L_{k-ij}}[m,n] \tag{2.64}$$

$$= \sum_{k=1}^{3}\sum_{m=1}^{3}\sum_{n=1}^{3} \boldsymbol{X_k}[m+i-1, n+j-1] \cdot \boldsymbol{W_k}[m,n]$$

$$\boldsymbol{H} = f(\boldsymbol{S}) \tag{2.65}$$

Notice that the convolution of 8x8x3 dimensional input $\boldsymbol{X}$ with 3x3x3 dimensional filter $\boldsymbol{W}$ produced 6x6 dimensional output matrix $\boldsymbol{H}$. In some cases, it could be desirable to keep the width and height of input and output of a convolutional layer same. In order to achieve 8x8 output matrix, the input could be padded with zeros at the borders as shown in Figure 2.24 before the CONV operation. Assuming that each border is padded symmetrically with P zeros (e.g. $P = 1$ in Figure 2.24) the convolution of W_1xH_1xN dimensional input $\boldsymbol{X}$ with a F_1xF_2xN dimensional filter $\boldsymbol{W}$ with vertical stride S_1 and horizontal stride S_2 will produce a W_2xH_2 dimensional matrix $\boldsymbol{H}$ with width $(W - F_1 + 2P)/S_1 + 1$ and height $(H - F_2 + 2P)/S_2 + 1$.

In general, the convolution operation is performed with a set of K independent F_1xF_2xN dimensional filters instead of one filter. This will produce a total of K independent W_2xH_2 dimensional matrices. These matrices are then stacked together to produce W_2xH_2xK dimensional output matrix $\boldsymbol{H}$.

(ii) Pooling (Pool) layer performs a sub-sampling operation on the output of a CONV layer to ensure shift and distortion invariance to some degree as stated by Lecun *et al.* [6]. Unlike CONV layers, activation functions are not used in the Pool layers. Moreover, the number of input channels stays the same after the pooling operation In other words, the pooling operation performed on a W_1xH_1xK dimensional matrix produces a W_2xH_2xK dimensional matrix as the output. For example, we could use a 2x2 filter $\boldsymbol{W}$ with horizontal and vertical stride 2 to down-sample a matrix $\boldsymbol{H_1}$ by two as shown in Figure 2.25, to produce $\boldsymbol{H_2}$ (i.e. the output of the pooling layer). In this example, we have used max-pooling (MaxPool) operation as a filter, which gives the maximum of its four inputs as the output.

X_1

0	0	0	0	0	0	0	0	0	0
0	10	4	3	24	44	42	55	21	0
0	20	6	1	5	88	233	67	89	0
0	50	12	7	66	85	33	14	1	0
0	40	13	11	44	63	53	43	2	0
0	50	55	73	90	97	3	1	4	0
0	30	11	62	64	57	5	5	2	0
0	80	24	24	85	5	0	8	6	0
0	10	255	10	20	20	40	50	70	0
0	0	0	0	0	0	0	0	0	0

X_2

0	0	0	0	0	0	0	0	0	0
0	2	1	5	0	2	4	33	4	0
0	3	3	0	1	3	3	6	46	0
0	5	4	7	55	54	22	1	1	0
0	77	8	1	55	44	79	22	4	0
0	98	1	44	12	64	56	0	0	0
0	64	5	99	0	66	46	0	64	0
0	84	9	12	1	55	95	22	4	0
0	63	73	150	244	60	80	100	200	0
0	0	0	0	0	0	0	0	0	0

X_3

0	0	0	0	0	0	0	0	0	0
0	3	4	4	3	3	1	2	1	0
0	1	0	2	0	1	0	4	1	0
0	4	2	4	1	7	5	8	9	0
0	0	1	4	4	2	2	1	8	0
0	7	4	6	6	2	1	2	8	0
0	6	3	6	2	5	2	2	8	0
0	2	1	3	3	7	5	1	7	0
0	1	6	9	0	7	5	0	6	0
0	0	0	0	0	0	0	0	0	0

Figure 2.24. Zero padded input matrix $\boldsymbol{X}$

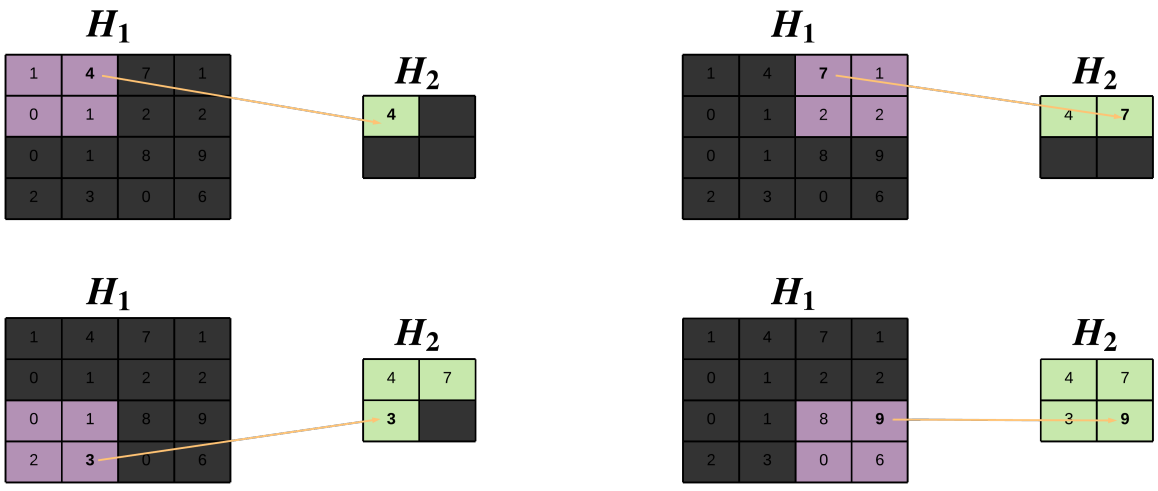

Figure 2.25. Max-pooling operation performed on matrix $\boldsymbol{H}$

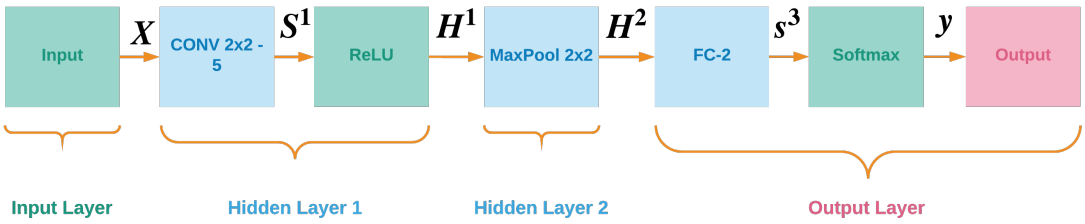

Figure 2.26. A convolutional neural network architecture with two hidden layers

Different CNN architectures are constructed by successively stacking CONV and Pool layers until the output layer. For example, Figure 2.26 shows a two layer CNN, where the convolution operation with F_1xF_2xNxK filters is represented with "$CONV\ F_1xF_2-K$" and the max pooling operation with F_1xF_2 filter is represented with "$Max\text{-}Pool\ F_1xF_2$". We have discarded N from the representation of the convolution operation since it can be inferred from the dimension of its input.

2.2.5. Recurrent Neural Networks

Recurrent neural networks (RNNs) are used to preserve the sequential information in the input space. In order to show this explicitly, we assume that our dataset contains N sequences $\{\mathcal{X}_1, \mathcal{X}_1, \ldots, \mathcal{X}_N\}$ and each of $\mathcal{X}_i$ contains $|\mathcal{X}_i|$ elements such that $\mathcal{X}_i = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x}_{|\boldsymbol{\mathcal{X}_i}|}\}$. For example, each of $\mathcal{X}_i$ could be a movie review (e.g. "watching this movie was not a mistake") which consists of a sequence of words $\boldsymbol{x_i}$ (e.g. "movie").

In order to process each review in a computer, each word in the dataset should be represented with a vector (i.e. *word embedding*). Initially, we should determine the number of *distinct* words we want to represent. The set which contains all the words we want to represent is called *vocabulary*. We could use a distinct word embedding for either each word in the training set or each word in a subset of the training set. In a simple way, we could count the number of times each word is used in the training set. Then, we could create this subset from the words with count more than some fixed number (e.g. three). In both cases, it is possible to encounter some words which are not in the vocabulary. For instance, the test set could contain some words which are not present in the training set. An extra embedding vector $< UNK >$ is used for such words.

After the vocabulary size $|V|$ (i.e. the number of distinct words we want to represent plus one for $< UNK >$) is determined, either *one-hot encoding* or *variable-sized encoding* is used to represent each element in the vocabulary with a vector $\boldsymbol{v_i}$ for $1 \leq i \leq |V|$. In the former method, each element is represented with a sparse vector of size $|V|$, such that only i^{th} element of the vector is one (i.e. $\boldsymbol{v_i}[i] = 1$) and all the other elements of the vector is zero (e.g. $\boldsymbol{v_i}[j] = 0$ for $1 \leq j \leq |V|$ and $j \neq i$). This method does not preserve the semantic similarity between the words since each element of $\boldsymbol{v_i}$ is an independent scalar.

On the other hand, the latter method encodes each element in the vocabulary with a dense vector of size $K < |V|$ such that semantically similar words have similar entries on the same vector positions. More concretely, a fully connected layer with the activation function $f(x) = x$ and $|V|xK$ dimensional weight matrix $\boldsymbol{W}$ is used for the *variable-sized encoding*. Each row of the weight matrix represents an embedding vector such that $\boldsymbol{v_i} = \boldsymbol{W}[i, :]$. The weight matrix is optimized during the training with the gradient descent to preserve the semantic information in the vocabulary.

After the number of elements in the vocabulary and the encoding method is decided, our objective is to classify each review $\mathcal{X}_i$ with either "good" or "bad" label y_i (e.g. "good" for this example). The length of each review is not the same, so we cannot use a FC layer which requires a fixed size input. RNNs process each word in a review sequentially in order to use a variable sized input. In other words, $\boldsymbol{x_1}$ is used as the input at the first step, $\boldsymbol{x_2}$ is used as the input at the second step and so on. Additionally, the outcome of the classification should depend not only on each word but also on their alignments. At each step of RNN, the information in the present step (i.e. $\boldsymbol{x_t}$) is combined with the information coming from the previous steps (i.e. $\boldsymbol{h_{t-1}}$) to produce the vector $\boldsymbol{h_t}$. Then, $\boldsymbol{h_t}$ is passed to the next step and the whole process is repeated recursively. On an intuitive level, $\boldsymbol{h_t}$ stores word alignment information of the sequence $\{\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{...}, \boldsymbol{x_t}\}$ and passes it through the next step. More rigorously, Equations (2.66) and (2.67) are used for the computation of $\boldsymbol{h_t}$ at t^{th} step, where $\boldsymbol{W_x}$ is the weight matrix of $\boldsymbol{x}$, $\boldsymbol{W_h}$ is the weight matrix of $\boldsymbol{h}$, $\boldsymbol{b}$ is the bias vector and $f()$ is the activation function. Same weight matrices and bias vector are used at every step. Additionally, $\boldsymbol{h_0}$ stores the prior knowledge about the sequence. We could use either a constant vector (e.g. a matrix whose each element is zero) for $\boldsymbol{h_0}$ or treat it as a parameter of the network and learn it during the training.

$$\boldsymbol{s_t} = \boldsymbol{W_x}\,\boldsymbol{x_t} + \boldsymbol{W_h}\,\boldsymbol{h_{t-1}} + \boldsymbol{b} \tag{2.66}$$

$$\boldsymbol{h_t} = f(\boldsymbol{s_t}) \tag{2.67}$$

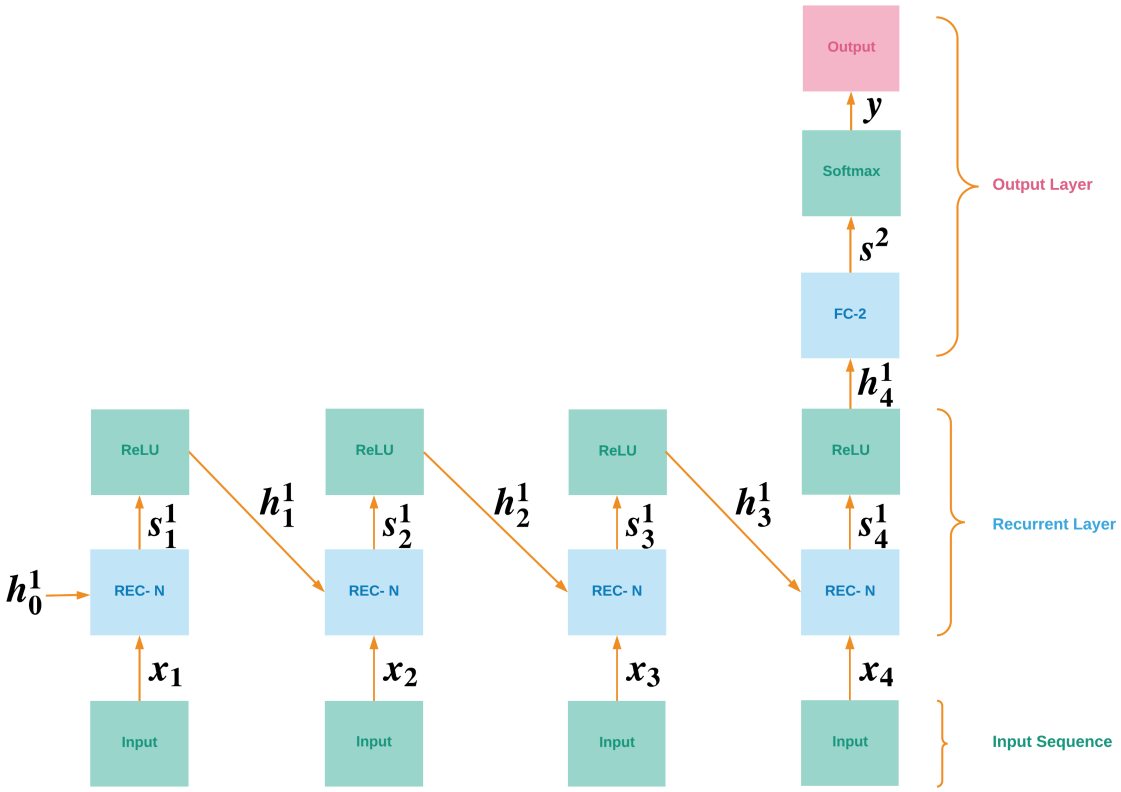

Figure 2.27. A RNN architecture designed to classify movie reviews as either good or bad

Figure 2.27 illustrates a RNN structure which is designed to classify a movie review $\mathcal{X} = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}, \boldsymbol{x_4}\}$ with either "good" or "bad" label y. We have used *REC-N* to represent a recurrent layer with N hidden neurons. Since there are only two output classes, *FC-2* layer is used at the output layer.

We could also design a RNN structure to predict the next word $\boldsymbol{w_{t+1}}$ in a sentence $\mathcal{W}$ based on all the previous words $\{\boldsymbol{w_1}, \boldsymbol{w_2}, \ldots, \boldsymbol{w_t}\}$. In practice, a sentence $\mathcal{W}$ which contains T words such that $\mathcal{W} = \{\boldsymbol{w_1}, \boldsymbol{w_2}, \ldots, \boldsymbol{w_T}\}$ is represented with $T+2$ embedding vectors $\widehat{\mathcal{W}} = \{< SOS >, \boldsymbol{w_1}, \boldsymbol{w_2}, \ldots, \boldsymbol{w_T}, < EOS >\}$, where $< SOS >$ and $< EOS >$ symbols indicate *the start of a sentence* and *the end of a sentence* respectively. In order to represent both $< SOS >$ and $< EOS >$ symbols with embedding vectors, the vocabulary size $|V|$ is increased by two. The final vocabulary includes all the distinct words we want to represent together with $< UNK >$, $< SOS >$ and $< EOS >$.

In the training phase, our objective is to predict the next word with the current word embedding and the hidden layer output coming from the previous step. As a consequence, all the embedding vectors except $< EOS >$ could be used as an input for the training. At each step of RNN, the expected output will be the embedding vector in $\widehat{\mathcal{W}}$ coming after the input embedding vector. For instance, $< SOS >$ is used to predict the first word of the sentence (i.e. $\boldsymbol{w_1}$) at the first step. $\boldsymbol{w_1}$ is used to predict the second word of the sentence (i.e. $\boldsymbol{w_2}$) at the second step and so on until the last word of the sentence (i.e. $\boldsymbol{w_T}$) is used to predict $< EOS >$. More concretely, our objective is to find the mapping between the input sequence $\mathcal{X} = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_{T+1}}\}$ and the output sequence $\mathcal{Y} = \{\boldsymbol{y_1}, \boldsymbol{y_2}, \ldots, \boldsymbol{y_{T+1}}\}$ for a sentence $\mathcal{W} = \{\boldsymbol{w_1}, \boldsymbol{w_2}, \ldots, \boldsymbol{w_T}\}$, where $\mathcal{X}$ and $\mathcal{Y}$ are given in Equations (2.68) and (2.69). Additionally, we have used the softmax function at the output layer to obtain the conditional probabilities of each embedding vector $\boldsymbol{v_j}$. Each dimension of the output vector gives the conditional probability of a different embedding vector as shown in Equation (2.70).

$$\mathcal{X} = \{< SOS >, \boldsymbol{w_1}, \boldsymbol{w_2}, \ldots, \boldsymbol{w_T}\} \tag{2.68}$$

$$\mathcal{Y} = \{\boldsymbol{w_1}, \boldsymbol{w_2}, \ldots, \boldsymbol{w_T}, < EOS >\} \tag{2.69}$$

$$p(\boldsymbol{v_j}|\boldsymbol{x_i}) = \boldsymbol{y_i}[j] \quad \text{for} \quad i \in \{1, 2, \ldots, T+1\} \\ j \in \{1, 2, \ldots, |V|\} \\ \boldsymbol{x_i} \in \boldsymbol{R}^K, \boldsymbol{y_i} \in \boldsymbol{R}^{|V|} \tag{2.70}$$

In the test (i.e. *inference*) phase, our objective is to predict the whole sentence $\mathcal{W} = \{\boldsymbol{w_1}, \boldsymbol{w_2}, \ldots, \boldsymbol{w_T}\}$. However, we only know that all sentences start with $< SOS >$ and end with $< EOS >$. Assuming that $\boldsymbol{h_0}$ stores the prior knowledge about the sentence $\mathcal{W}$, there are three main methods to generate this sentence.

(i) Greedy Search

$\boldsymbol{x_1}$ (i.e. the embedding vector of $< SOS >$ symbol) is used to compute $\boldsymbol{y_1}$ at the first step. The embedding vector with the highest conditional probability is chosen as the first word of the sentence (i.e. $\widehat{\boldsymbol{w_1}}$). Then, $\widehat{\boldsymbol{w_1}}$ is used as the input at the second step (i.e. as $\boldsymbol{x_2}$) to compute $\boldsymbol{y_2}$. Similar to the previous step, the embedding vector with the highest conditional probability is chosen as the second word of the sentence (i.e. $\widehat{\boldsymbol{w_2}}$). This process is repeated recursively to generate a sentence until $< EOS >$ is chosen as the word embedding with the highest conditional probability. The maximum number of recursive steps are fixed to ensure that the generation process terminates eventually.

(ii) Beam Search [26]

Similar to *Greedy Search*, $\boldsymbol{x_1}$ (i.e. the embedding vector of $< SOS >$ symbol) is used as input to compute $\boldsymbol{y_1}$ at the first step. Instead of choosing a single word at this step, β words (i.e. also called *beam width*) with the highest conditional probabilities are chosen as *candidate words* for the first step. Then, each candidate word from the previous step is used as the input at the second step (i.e. as $\boldsymbol{x_2}$) to obtain β *candidate words*. Since we are using β candidates from the previous step, there are a total of β^2 candidate words for the second step. Considering $\boldsymbol{v_i}$ as one of β candidates at the first step and $\boldsymbol{v_j}$ as one of β candidates at the second step

for $\boldsymbol{x_2} = \boldsymbol{v_i}$, then the conditional probability of two-word sequence $\{\boldsymbol{v_i}, \boldsymbol{v_j}\}$ could be calculated with Equation (2.71). Since our objective is to generate the word sequence with the highest probability, we choose β two-word sequences with the highest conditional probabilities as *candidate two-word sequences* at the second step. This process is repeated recursively T times to obtain β *candidate T-word sequences*. The predicted sentence is chosen as the candidate sequence with the highest conditional probability.

$$\begin{aligned} p(\boldsymbol{v_i}, \boldsymbol{v_j} | \boldsymbol{x_1}, \boldsymbol{x_2}) &= p(\boldsymbol{v_j} | \boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{v_i}) * p(\boldsymbol{v_i} | \boldsymbol{x_1}) \\ &= \boldsymbol{y_2}[j] * \boldsymbol{y_1}[i] \end{aligned} \tag{2.71}$$

(iii) Maximum Likelihood Estimation

Since our objective is to generate the sentence with the highest probability $p(\widehat{\mathcal{W}}|\mathcal{X})$ we could calculate the conditional probability for all $|V|^T$ word sequences to generate a sentence with T words. Although this method will give us the most likely sentence, it is computationally expensive than both *Greedy Search* and *Beam Search*.

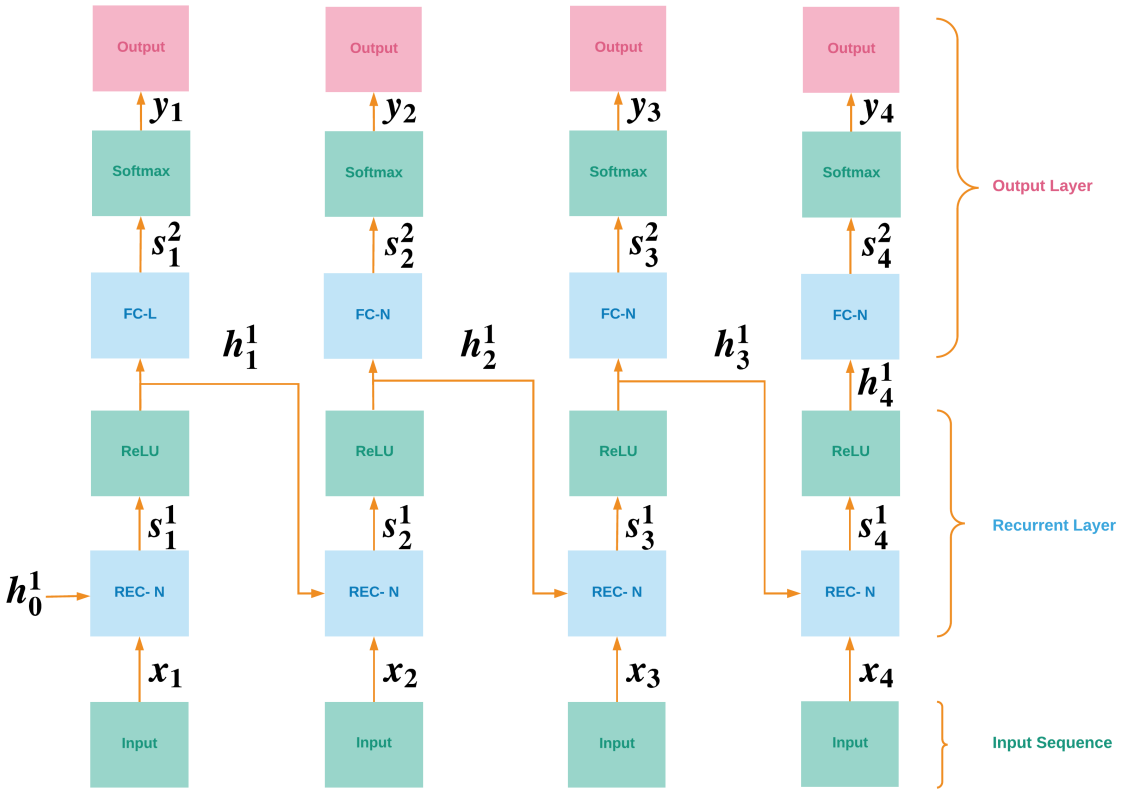

Figure 2.28. RNN diagram designed for the sentence generation

Although RNNs could store some form of sequential information, Bengio *et al.* [47] showed that they are not suitable for preserving long-term dependencies in a sequence. In particular, the gradient of the loss with respect to the weight matrices could get either very large (i.e. exploding gradient) or very small (i.e. vanishing gradient) during the optimization. In order to overcome these problems, Hochreiter and Schmidhuber [27] proposed *Long Short-Term Memory* (LSTM) Networks. Equations (2.72)-(2.77) are used for the computation of $\boldsymbol{h_t}$ at t^{th} step, where σ is the sigmoid function and $\circ$ is the element-wise multiplication.

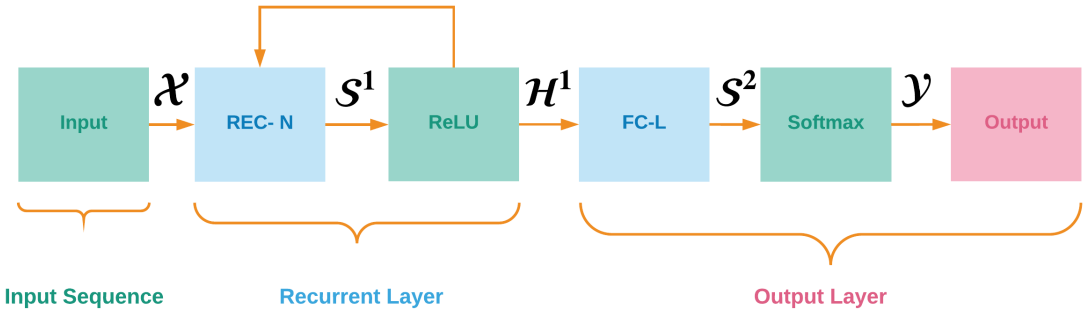

Figure 2.29. Another way to show RNN diagram designed for the sentence generation

$$\boldsymbol{i_t} = \sigma(\boldsymbol{W_{ix}}\,\boldsymbol{x_t} + \boldsymbol{W_{ih}}\,\boldsymbol{h_{t-1}} + \boldsymbol{b_i}) \tag{2.72}$$

$$\boldsymbol{f_t} = \sigma(\boldsymbol{W_{fx}}\,\boldsymbol{x_t} + \boldsymbol{W_{fh}}\,\boldsymbol{h_{t-1}} + \boldsymbol{b_f}) \tag{2.73}$$

$$\boldsymbol{o_t} = \sigma(\boldsymbol{W_{ox}}\,\boldsymbol{x_t} + \boldsymbol{W_{oh}}\,\boldsymbol{h_{t-1}} + \boldsymbol{b_o}) \tag{2.74}$$

$$\boldsymbol{g_t} = tanh(\boldsymbol{W_{gx}}\,\boldsymbol{x_t} + \boldsymbol{W_{gh}}\,\boldsymbol{h_{t-1}} + \boldsymbol{b_g}) \tag{2.75}$$

$$\boldsymbol{c_t} = \boldsymbol{f_t} \circ \boldsymbol{c_{t-1}} + \boldsymbol{i_t} \circ \boldsymbol{g_t} \tag{2.76}$$

$$\boldsymbol{h_t} = \boldsymbol{o_t} \circ tanh(\boldsymbol{c_t}) \tag{2.77}$$

Note that, $\boldsymbol{c_t}$ (i.e. *cell state*) stores the alignment information of the sequence $\{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_t}\}$ and passes it through the next step (i.e. *next cell*) in LSTM Networks. $\boldsymbol{f_t} \circ \boldsymbol{c_{t-1}}$ represents the information coming from the previous cell. $\boldsymbol{f_t}$ (i.e. *forget gate*) modifies the previous cell state $\boldsymbol{c_{t-1}}$ to determine how much of the previous information should be forgotten. $\boldsymbol{i_t} \circ \boldsymbol{g_t}$ represents the amount of information which is added to $\boldsymbol{f_t} \circ \boldsymbol{c_{t-1}}$ at the current cell. $\boldsymbol{i_t}$ (i.e. *input gate*) modifies $\boldsymbol{g_t}$ with a value in range $[0, 1]$. After the cell state is calculated, it is passed through *tanh* activation function and modified with $\boldsymbol{o_t}$ (i.e. *output gate*) to produce the cell output $\boldsymbol{h_t}$. The whole process is repeated recursively.

2.2.6. Batch Normalization Layer

Each layer of a neural network learns the mapping between its input and output distribution. However, the input distribution of a layer changes as the model parameters are updated during the training. In other words, this layer experiences *covariance shift* [48], which increases the time required for the optimization. Ioffe and Szegedy [49] showed that the normalization of layer inputs reduces the severity of this problem. In order to achieve this, each dimension of the layer input $\mathbf{x}$ is initially normalized to have zero means and unit variance using Equation (2.78), where $\mathbf{x}$ is a d dimensional vector, $\boldsymbol{x}[k]$ is k^{th} dimension of $\boldsymbol{x}$ and both expectation and variance are calculated in the training set $\mathcal{D}$. Then, the linear transformation given in Equation (2.79) is applied to each $\widehat{\boldsymbol{x}}[k]$, where $\boldsymbol{y}$ is the batch normalization layer output, ϵ is a small number which is added to the denominator for numerical stability and both $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are d dimensional vectors which are learned during the training.

$$\widehat{\boldsymbol{x}}[k] = \frac{\boldsymbol{x}[k] - E(\boldsymbol{x}[k])}{\sqrt{Var(\boldsymbol{x}[k] + \epsilon)}} \tag{2.78}$$

$$\boldsymbol{y}[k] = \boldsymbol{\gamma}[k]\, \widehat{\boldsymbol{x}}[k] + \boldsymbol{\beta}[k] \tag{2.79}$$

Using gradient descent in the entire dataset $\mathcal{D}$ could be slow and infeasible because of high GPU memory requirement. Instead we could apply mini-batch gradient descent by splitting the training set into N mutually exclusive and collectively exhaustive sets (i.e. *mini-batch*) each of which has m samples and updating the model parameters for each *mini-batch*. Assuming we have a mini-batch $\mathcal{B}$ with m samples, both the expectation and variance in Equation (2.78) are calculated for each mini-batch $\mathcal{B}$ instead of whole training dataset $\mathcal{D}$. More concretely, consider a batch normalization layer with m inputs $\{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}\}$ each of which has d dimension. Following steps are applied in the training phase to calculate m batch normalization layer outputs

$\{\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_m}\}$ each of which has also d dimension.

(i) The mean and variance for each dimension of $\mathbf{x}$ in the mini-batch is calculated.

$$\boldsymbol{\mu}_{\mathcal{B}}[k] = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x_i}[k] \tag{2.80}$$

$$\boldsymbol{\sigma^2}_{\mathcal{B}}[k] = \frac{1}{m}\sum_{i=1}^{m} (\boldsymbol{x_i}[k] - \boldsymbol{\mu}_{\mathcal{B}}[k])^2 \tag{2.81}$$

(ii) Each dimension of $\mathbf{x}$ is normalized to have zero mean and unit variance.

$$\widehat{\boldsymbol{x}}_{\boldsymbol{i}}[k] = \frac{\boldsymbol{x_i}[k] - \boldsymbol{\mu}_{\mathcal{B}}[k]}{\sqrt{\boldsymbol{\sigma^2}_{\mathcal{B}}[k] + \epsilon}} \tag{2.82}$$

(iii) Linear transformation with parameters $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ is applied to the normalized inputs. These parameters are updated during the training with the other model parameters using mini-batch gradient descent.

$$\boldsymbol{y_i}[k] = \boldsymbol{\gamma}[k]\, \widehat{\boldsymbol{x}}_{\boldsymbol{i}}[k] + \boldsymbol{\beta}[k] \tag{2.83}$$

During the test phase, the output of a neural network should depend only on the input in order to get the same output every time we use the same input, since we want to build a deterministic model not a stochastic one. For this reason, the mean and the variance for each dimension of $\mathbf{x}$ are calculated over mini-batches during the training phase as shown in Equations (2.84) and (2.85). Then, the resulting mean and variance is inserted into Equations (2.78) and (2.79) to calculate batch normalization layer output during the test phase.

$$E(\boldsymbol{x}[k]) = E_{\mathcal{B}}(\boldsymbol{\mu}_{\mathcal{B}}[k]) \tag{2.84}$$

$$Var(\boldsymbol{x}[k]) = \frac{m}{m-1} E_{\mathcal{B}}(\boldsymbol{\sigma^2}_{\mathcal{B}}[k]) \tag{2.85}$$

2.2.7. Dropout

Dropout [35, 50] is a popular regularization technique which is used to increase the generalization probability of neural networks. Specifically, each neuron in a layer is independently set to zero with probability $p_{do} \in [0, 1]$ during only the training phase. This operation is repeated for each minibatch so that different neurons are set to zero for different minibatches.

3. REGULARIZING DESCRIPTION GENERATION MODEL

In this section, we describe our main model, which is used for generating image descriptions. Our objective is to design a model, which takes an image as the input and generates a sentence at its output as shown in Figures 3.1. More concretely, our model uses a Convolutional Neural Network architecture (see Section 2.2.4) to extract the image information. Then, a Long Short Time Memory Network (see Section 2.2.5) uses this information to generate a sentence.

Figure 3.1. An overview of our description generation model, which takes an image as input and generates a sentence at its output

In the training phase, we use both images and their descriptions to train our model. However, our objective is to generate these descriptions with using only images in the test (i.e. *inference*) phase. Because of this reason, our model behaves differently in the training and inference phase similar to the sentence generation model described in Section 2.2.5. We will explain both the training phase and the inference phase behavior of our model in the following sections.

3.1. Description Generation Model

3.1.1. CNN Architecture

Our objective is to design a CNN architecture to extract the image information as a fixed size vector. In order to reduce the *over-fitting*, we employ the data augmentation techniques used in [5, 7, 9]. More concretely, each image is resized into 256x256x3 size initially. Then, a 224x224x3 dimensional crop is randomly sampled from either the resized image or its horizontal flip. We extract the per-channel mean from each pixel and divide the resulting value by the per-channel standard deviation to normalize the random crop. Mean and standard deviation values are calculated for each channel only in ImageNet training set [51]. However, they are used for the normalization of the pixels in the same channel during both the training and inference phase. The resulting 224x224x3 dimensional image is used as CNN input.

As the CNN architecture, we use 152 layer CNN structure (referred as *ResNet-152*) proposed by He *et al.* [9], which won the 1^{st} place in the ILSVRC 2015 image classification competition [25], to extract image information. However, the last layer of *ResNet-152* uses a fully connected layer with 1000 dimensional output and the softmax activation function to calculate the conditional probabilities of 1000 classes present in ImageNet. Instead, we use a fully connected layer with 512 dimensional output followed by a batch normalization layer [49] (see Section 2.2.6) to calculate the image information. We rename the resulting CNN architecture as *modified ResNet-152* and use it to calculate a 256 dimensional vector for each image.

In order to train the *modified ResNet-152*, we are not using any explicit labels. Instead, we will train the *modified ResNet-152* together with LSTM to learn the network parameters. These parameters could be initialized randomly at the start of the training phase. Alternatively, several works [52–54] showed that instead of initializing the weights of a neural network randomly for one task, we could initially train a similar neural network on a different dataset for a different task and then use the weights of the trained neural network for parameter initialization. This idea is referred as *transfer*

learning [55–57] and arises from the fact that the initial layers of the neural networks which are trained on natural images learn task and dataset independent features such as Gabor filters or color blobs as stated by Yosinski *et al.* [54] and shown in the work of Zeiler and Fergus [58]. Following this approach, we use the weights of *ResNet-152* trained on ImageNet for image classification in order to initialize the weights of all but the last two layers of *modified ResNet-152*. The last FC layer is initialized with the weights drawn from a normal distribution with mean 0 and standard deviation 0.02.

We could optimize either all the transferred layers or some of them together with the last two layers. Although these approaches increases the performance as shown in the work of Yosinski *et al.* [54], they require high GPU memory during the training to keep both the activations and the gradients of each layer in the memory. As a consequence, we optimize only the last two layers of the *modified ResNet-152* during the training and use the transferred weights without any change to fit our description generation model to GPU memory.

3.1.2. LSTM Architecture

We use all unique words in the training set together with $< unk >$ and $< eos >$ tokens to construct vocabulary $\mathcal{V}$ with size of $|\mathcal{V}|$. $< unk >$ token represents words that are not in the vocabulary, while $< eos >$ token indicates the end of a sentence. Each token in this vocabulary is represented with a h dimensional embedding vector. The embedding vector of i-th token in $\mathcal{V}$ is computed using the equation $\boldsymbol{x_i} = \boldsymbol{W_\mathcal{V}}\ \boldsymbol{I_i}$, where $\boldsymbol{x_i}$ is the embedding vector, $\boldsymbol{W_\mathcal{V}}$ is the word embedding matrix with dimensions h x $|\mathcal{V}|$ and $\boldsymbol{I_i}$ is a sparse column vector of size $|\mathcal{V}|$, such that only i-th element of $\boldsymbol{I_i}$ is one and all the others are zero.

Similar to the work of Vinyals *et al.* [3], we use a LSTM architecture (see Section 2.2.5) to obtain the probability distribution of a word in a sentence given the previous words and the image $\boldsymbol{I}$. Formally, the probability distribution of $(t+1)$-th word $\boldsymbol{y_t} = \boldsymbol{p}(\boldsymbol{w_{t+1}}|\boldsymbol{I}, \boldsymbol{w_1}, \ldots, \boldsymbol{w_t})$ is calculated using the following formulations:

$$\boldsymbol{x_0} = CNN(\boldsymbol{I}) \tag{3.1}$$

$$\boldsymbol{i_t} = \sigma(\boldsymbol{W_{ix}}\,\boldsymbol{x_t} + \boldsymbol{W_{ih}}\,\boldsymbol{h_{t-1}} + \boldsymbol{b_i}) \tag{3.2}$$

$$\boldsymbol{f_t} = \sigma(\boldsymbol{W_{fx}}\,\boldsymbol{x_t} + \boldsymbol{W_{fh}}\,\boldsymbol{h_{t-1}} + \boldsymbol{b_f}) \tag{3.3}$$

$$\boldsymbol{o_t} = \sigma(\boldsymbol{W_{ox}}\,\boldsymbol{x_t} + \boldsymbol{W_{oh}}\,\boldsymbol{h_{t-1}} + \boldsymbol{b_o}) \tag{3.4}$$

$$\boldsymbol{g_t} = tanh(\boldsymbol{W_{gx}}\,\boldsymbol{x_t} + \boldsymbol{W_{gh}}\,\boldsymbol{h_{t-1}} + \boldsymbol{b_g}) \tag{3.5}$$

$$\boldsymbol{c_t} = \boldsymbol{f_t} \circ \boldsymbol{c_{t-1}} + \boldsymbol{i_t} \circ \boldsymbol{g_t} \tag{3.6}$$

$$\boldsymbol{h_t} = \boldsymbol{o_t} \circ tanh(\boldsymbol{c_t}) \tag{3.7}$$

$$\boldsymbol{y_t} = softmax(\boldsymbol{W_p}\,\boldsymbol{h_t} + \boldsymbol{b_p}) \tag{3.8}$$

where $[\boldsymbol{W_{ix}}, \boldsymbol{W_{ih}}, \boldsymbol{W_{fx}}, \boldsymbol{W_{fh}}, \boldsymbol{W_{ox}}, \boldsymbol{W_{oh}}, \boldsymbol{W_{gx}}, \boldsymbol{W_{gh}}]$ are LSTM weight matrices, $[\boldsymbol{b_i}, \boldsymbol{b_f}, \boldsymbol{b_o}, \boldsymbol{b_g}]$ are LSTM bias vectors, $\boldsymbol{c_t}$ is the vector storing cell state at timestep t, $\boldsymbol{h_t}$ is the vector storing hidden state at timestep t, $\circ$ is the element-wise multiplication, $\boldsymbol{w_t}$ is the t-th word in a sentence, $\boldsymbol{x_t}$ is the embedding vector of $\boldsymbol{w_t}$ for $t \in \{1, \ldots, T\}$ and $\boldsymbol{x_0}$ is the *modified ResNet-152* output which is used as 0-th word of sentence representing a-priori information about the sentence. In these equations, $\boldsymbol{c_t}$ (i.e. *cell state*) stores the alignment information of the sequence $\{\boldsymbol{x_0}, \boldsymbol{x_1}, \ldots, \boldsymbol{x_t}\}$ and passes it through the next step (i.e. *next cell*) in LSTM Networks. $\boldsymbol{f_t} \circ \boldsymbol{c_{t-1}}$ represents the information coming from the previous cell. $\boldsymbol{f_t}$ (i.e. *forget gate*) modifies the previous cell state $\boldsymbol{c_{t-1}}$ to determine how much of the previous information should be forgotten. $\boldsymbol{i_t} \circ \boldsymbol{g_t}$ represents the amount of information which is added to $\boldsymbol{f_t} \circ \boldsymbol{c_{t-1}}$ at the current cell. $\boldsymbol{i_t}$ (i.e. *input gate*) modifies $\boldsymbol{g_t}$ with a value in range $[0, 1]$. After the cell state is calculated, it is passed through *tanh* activation function and modified with $\boldsymbol{o_t}$ (i.e. *output gate*) to produce the cell output $\boldsymbol{h_t}$.Then, $\boldsymbol{h_t}$ is passed through a linear layer with weight matrix $\boldsymbol{W_p}$ and bias vector $\boldsymbol{b_p}$ together with a softmax activation function to obtain $\boldsymbol{y_t}$.

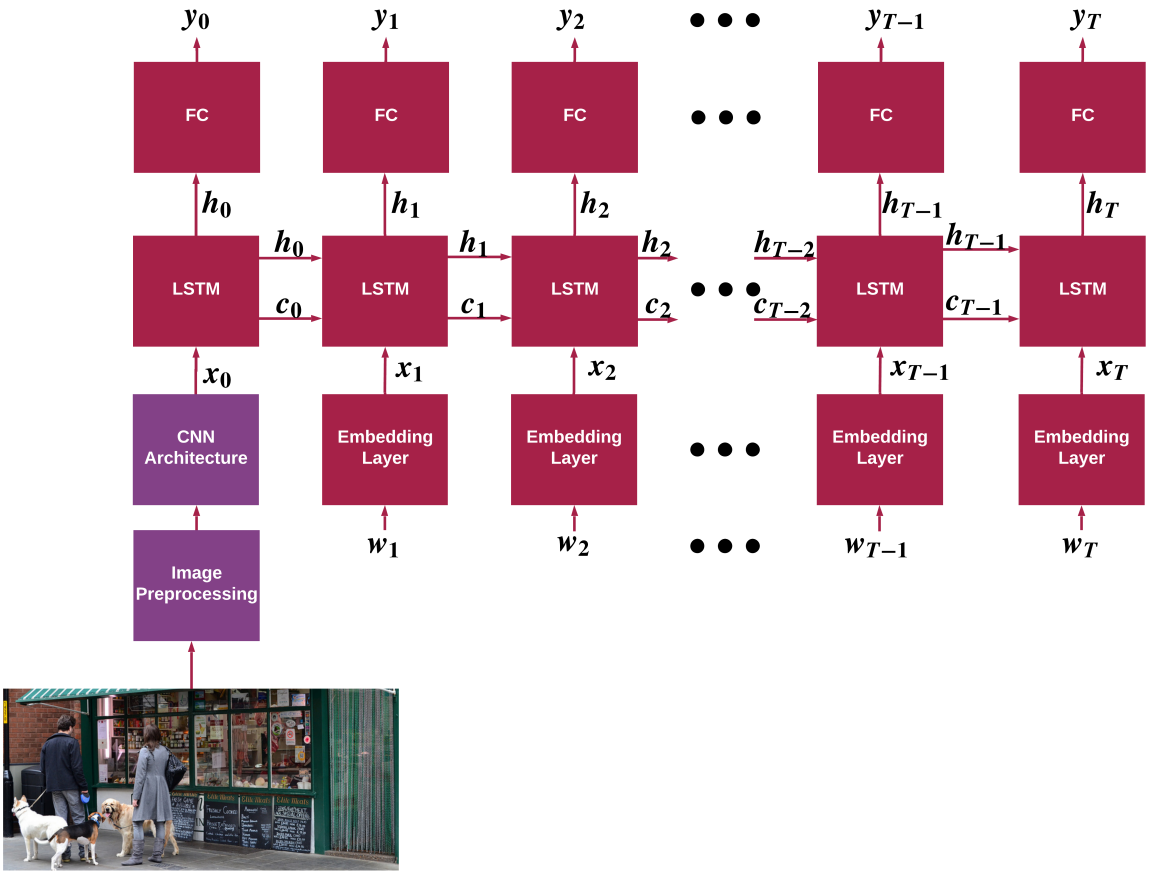

Figure 3.2. Training phase of our image captioning model

Training phase. The training phase of our image captioning model is shown in Figure 3.2. LSTM architecture is trained to maximize the probability of the correct word given the previous words and an image. Initially, we set both $\boldsymbol{c_{-1}}$ and $\boldsymbol{h_{-1}}$ to zero and use the image information $\boldsymbol{x_0}$ as input to obtain the probability distribution of first word, $\boldsymbol{y_0}$. Then, the embedding vector of first word $\boldsymbol{x_1}$ is used together with $\boldsymbol{c_0}$ and $\boldsymbol{h_0}$ to obtain the probability distribution of second word $\boldsymbol{y_1}$ and so on. On the last step, we maximize the probability of $< eos >$ token using $\boldsymbol{x_T}$, $\boldsymbol{c_{T-1}}$ and $\boldsymbol{h_{T-1}}$ as inputs.

Inference phase. Figure 3.3 illustrates the inference phase of our image captioning model, where the word with the highest probability at t-th step is represented with $\widehat{\boldsymbol{w}}_{\boldsymbol{t}}$. We aim to generate a sentence representing the contents of an image during this phase. Similar to training phase, initially we set both $\boldsymbol{c_{-1}}$ and $\boldsymbol{h_{-1}}$ to zero and

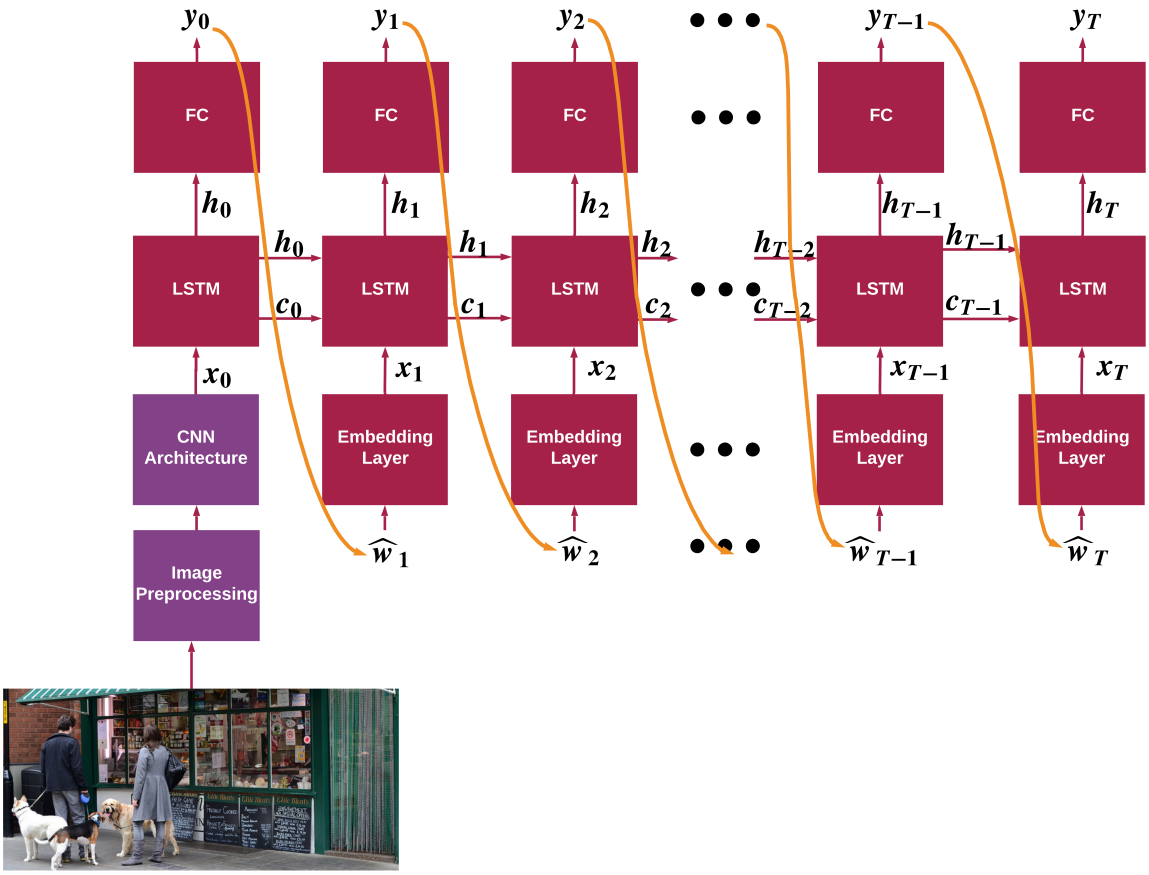

Figure 3.3. Inference phase of our image captioning model

use the image information $\boldsymbol{x_0}$ as input to obtain the probability distribution of first word, $\boldsymbol{y_0}$. We pick the word with the highest probability from $\boldsymbol{y_0}$, namely $\widehat{\boldsymbol{w}}_\mathbf{1}$ using greedy search (see Section 2.2.5) and use its embedding vector as input to obtain $\boldsymbol{y_1}$. Then, we repeat this process until we pick $< eos >$ token or a sentence with length of 20 is generated.

3.2. Regularization Description Generation Model

Merity *et al.* [30] used a set of regularization techniques to improve the performance of word-level language models. Inspired by their work, we used some of these techniques to improve the performance of our model. In this section, we explain these regularization techniques.

3.2.1. Weight-dropped LSTM

Dropconnect is introduced by Wan *et al.* [59] as a generalization of dropout. Unlike dropout, each weight before a layer is independently set to zero with probability $p \in [0, 1]$ during only the training phase. This operation is repeated for each minibatch so that different weights are set to zero for different minibatches.

Weight-dropped LSTM method suggests using dropconnect technique on hidden to hidden weight matrices $[\boldsymbol{W_{ih}}, \boldsymbol{W_{fh}}, \boldsymbol{W_{oh}}, \boldsymbol{W_{gh}}]$ in order to regularize LSTM networks. Dropconnect technique is applied to the weight matrices once at the start of each minibatch, so that the same weights remain zero for each time step within the same minibatch.

3.2.2. Variational Dropout

Variational dropout is proposed by Gal and Ghahramani [60] as a regularization method for recurrent neural networks. Dropout technique (see Section 2.2.7) is applied to each neuron in a layer at the start of each minibatch for variational dropout, so that the same neurons remain zero for each time step within the same minibatch. This operation is applied for each sample in a minibatch independently. Note that the difference between dropout and variational dropout is that we sample a random dropout mask for each time step within a minibatch for the former technique, whereas we use the same dropout mask for each time step within a minibatch for the latter one.

3.2.3. Embedding Dropout

Gal and Ghahramani [60] used embedding dropout as a regularization method for word embedding matrix $\boldsymbol{W_{\mathcal{V}}} \in R^{VxE}$, where V is the number of words in the vocabulary and E is the size of embedding vector. Assuming that embedding dropout is performed with probability p_e, each row of weight matrix is set to zero with probability p_e. After the dropout operation applied, the rows with non-zero entries are scaled by $\frac{1}{1-p_e}$ to keep the expected value of a word embedding constant. Embedding dropout enforces

model to give equal emphasis on each word, instead of relying on a few words to solve a problem. Considering our description generation model, this regularization technique is equivalent to erasing some words in a sentence randomly.

3.2.4. Weight Tying

The total number of parameters in a neural network affects the memory consumption during both the training and the inference phases. [61, 62] suggested using weight tying in language modeling, which not only decreases the memory consumption but also increases the generalization probability of a model. The core idea behind weight tying is using the same weights for both the embedding and output layer. Since most of the model parameters for our description generation model belongs to linear layers, applying weight tying reduces the total number of parameters in our model from 87 million to 74 million, 14.8% relative decrease in the total number of parameters. Only one embedding vector is updated per time step for a sample during the training phase of a recurrent neural network without weight tying. Introducing weight tying increases the generalization probability of a model and decreases the convergence time because all embedding vectors are updated at each time step thanks to the coupling between embedding and output layer.

3.2.5. Activation Regularization and Temporal Activation Regularization

As the number of parameters in a neural network increases, the probability of overfitting increases as well. L_2 regularization [35], which uses the norm of the weights as additional loss term, is generally used to decrease this probability. Instead of the weights, the works of [30, 63] used L_2 regularization on the hidden layer outputs and on the difference of consecutive hidden layer outputs of a recurrent neural network to improve their word level language model. These techniques are named as activation regularization (AR) and temporal activation regularization (TAR) respectively.

More technically, AR uses R_{AR} given in Equation 3.9 and TAR uses R_{TAR} given in Equation 3.10 as an additional loss term, where T is the total number of time steps, α and β are scaling coefficients, $\boldsymbol{h_t}$ is the hidden layer output at t^{th} step, $\circ$ is the element-wise multiplication operation and $\boldsymbol{d_t}$ is the dropout mask applied to $\boldsymbol{h_t}$.

$$R_{AR} = \frac{1}{T}\sum_{t=1}^{T} \alpha L_2(\boldsymbol{h_t} \circ \boldsymbol{d_t}) \tag{3.9}$$

$$R_{TAR} = \frac{1}{T-1}\sum_{t=1}^{T-1} \beta L_2(\boldsymbol{h_t} - \boldsymbol{h_{t+1}}) \tag{3.10}$$

4. DATASETS AND EXPERIMENTS

4.1. Datasets

We use MS COCO (i.e. *Microsoft Common Objects in Context*) dataset [64, 65] to evaluate the performance of our image captioning models. There are 82,783 images in training set, 40,504 images in validation set and 40,775 images in testing set. Each image is annotated with 5 different descriptions. Two of these images are shown with their descriptions in Figures 4.1 and 4.2. Human generated descriptions are only provided for training and validation sets. We train our model on MS COCO training set, use 20,000 images in MS COCO validation set as our validation set and use the remaining 20,514 images in MS COCO validation set as our test set.

Figure 4.1. An image from MS COCO dataset with descriptions; (i) a passenger train that is pulling into the station. (ii) a train engine carrying carts into a station. (iii) people watching a train arrive at a train station. (iv) a red and black train some people and tracks (v) there is a red train that is coming up the tracks

Figure 4.2. Another image from MS COCO dataset with descriptions; (i) a baseball player swinging a bat at a baseball game. (ii) the batter hits the ball while his opponents look on (iii) a baseball player swings a bat at a ball (iv) a baseball player hits a ball as the other team looks on. (v) a baseball player holding a bat on top of a field.

We use a small object detection dataset (i.e. retail dataset) to evaluate the performance of our zero shot object detector described in subsection 5.2. There are $1,249$ images in training set, 157 images in validation set and 155 images in testing set. Each image contains at least one cabinet, poster, pricecard, umbrella or wastebin. Accordingly, we annotate each image with at least one label. An image from retail dataset containing a cabinet, a poster, a pricecard and a wastebin is shown in Figure 4.3.

Figure 4.3. An image from retail dataset containing a cabinet, a poster, a price card and a waste bin

4.2. Evaluation Metrics

We use the evaluation server of Chen *et al.* [65] in order to calculate the evaluation metrics in MS COCO dataset. In this section, initially we explain these metrics, namely CIDEr [66], BLEU [67], ROUGE [68] and METEOR [69]. Then, we define the metrics used to measure the performance of our zero shot object detector, which is explained in Section 5.2, namely true positive (TP), false positive (FP), false negative (FN), true negative (TN), precision and recall.

Each image I_i in MS COCO has five human generated (i.e. reference) descriptions $S_i = \{s_{i1}, \ldots, s_{im}\} \in S$ and one model generated (i.e. candidate) description $c_i \in C$ for which we want to compute evaluation metrics. Each description is represented with n-grams, where an n-gram $w_k \in \Omega$ is a set of n ordered words and Ω is the vocabulary of all n-grams. All evaluation scores are calculated with n grams with one to four words.

The number of times w_k occurs in the model generated description c_i is denoted as $h_k(c_i)$, whereas the number of times it occurs in the human generated description s_{ij} is denoted as $h_k(s_{ij})$.

CIDEr [66] uses term frequency inverse document frequency (TF-IDF) weighting for each n gram w_k to predict the human consensus in generated image descriptions. Equation 4.1 is used to compute TF-IDF weights $g_k(s_{ij})$ of each w_k, where I is the set of all images in the data set and $|I|$ is the size of this set (i.e. the number of images).

$$g_k(s_{ij}) = \frac{h_k(s_{ij})}{\sum_{w_l \in \Omega} h_l(s_{ij})} \log\left(\frac{|I|}{\sum_{I_p \in I} min(1, \sum_q h_k(s_{pq}))}\right) \tag{4.1}$$

These TF-IDF weights are used to compute the score for n-grams of length n, CIDEr_n, between candidate sentence c_i and the reference sentences, S_i, for image I_i as shown in Equation 4.2, where $\boldsymbol{g^n}(c_i)$ is a vector created by concatenating $g_k(c_i)$ for all n-grams of length n and $\|\boldsymbol{g^n}(c_i)\|$ is its magnitude.

$$\text{CIDEr}_n(c_i, S_i) = \frac{1}{m} \sum_j \frac{\boldsymbol{g^n}(c_i) \cdot \boldsymbol{g^n}(s_{ij})}{\|\boldsymbol{g^n}(c_i)\| \, \|\boldsymbol{g^n}(s_{ij})\|} \tag{4.2}$$

CIDEr score is computed by combining CIDEr_n as given in Equation 4.3. However, CIDEr sometimes gives a high score to a candidate sentence which gets low scores from humans. Vedantam *et al.* [66] suggested a modification to CIDEr_n, named as CIDEr-D_n to solve this problem. Equation 4.3 shown the computation of CIDEr-D_n, where $l(c_i)$ is the length of the candidate sentence c_i, $l(s_{ij})$ is the length of the reference sentence s_{ij} and $\sigma = 6$.

$$\text{CIDEr}(c_i, S_i) = \frac{1}{4}\sum_{n=1}^{4}\text{CIDEr}_n(c_i, S_i) \tag{4.3}$$

$$\text{CIDEr-D}_n(c_i, S_i) = \frac{10}{m}\sum_{j}\exp(\frac{-(l(c_i)-l(s_{ij}))^2}{2\sigma^2})\cdot\frac{\min(\boldsymbol{g^n}(c_i), \boldsymbol{g^n}(s_{ij}))\cdot\boldsymbol{g^n}(s_{ij})}{\|\boldsymbol{g^n}(c_i)\|\,\|\boldsymbol{g^n}(s_{ij})\|} \tag{4.4}$$

CIDEr-D computed similar to CIDEr as shown in Equation 4.5. We use CIDEr-D instead of CIDEr in our evaluations because of its robustness.

$$\text{CIDEr-D}(c_i, S_i) = \frac{1}{4}\sum_{n=1}^{4}\text{CIDEr-D}_n(c_i, S_i) \tag{4.5}$$

BLEU [67] uses corpus level clipped n-gram precision, CP_n, between the candidate sentences and reference sentences to evaluate the performance. CP_n is correlated with the number of n-gram overlaps between the candidate and reference sentences as shown in Equation 4.6. However, it is easier for a small candidate sentence to get a high score with this metric. Brevity penalty given in Equation 4.7 is used to overcome this problem, where l_C is the total length of candidate sentences and l_S is the total length of reference sentences. When there are multiple human generated sentences $S_i = \{s_{i1}, \ldots, s_{im}\} \in S$, only the sentence s_{ij} whose length is the closest to the length of c_i is used for the calculation of l_S.

$$CP_n(C, S) = \frac{\sum_i \sum_j min\Big(h_k(c_i), \underset{j\in m}{max}\, h_k(s_{ij})\Big)}{\sum_i \sum_j h_k(c_i)} \tag{4.6}$$

$$b(C,S) = \begin{cases} 1, & \text{if } l_C > l_S \\ \exp(1 - l_C/l_S), & \text{if } l_C \leq l_S \end{cases} \tag{4.7}$$

BLEU score is calculated using Equation 4.8, where $N = 1, 2, 3, 4$. We use $BLEU_1$, $BLEU_2$, $BLEU_3$ and $BLEU_4$ in our evaluations, where (C, S) is dropped for brevity from now on .

$$BLEU_N(C,S) = b(C,S)\, exp\Big(\frac{1}{N}\sum_{n=1}^{N} \log CP_n(C,S)\Big) \tag{4.8}$$

ROUGUE [68] uses the largest common subsequence (LCS) between candidate sentences and reference sentences to measure the performance. An LCS is similar to n-grams in a way that they both search for sequential words in a sentence. However, two sentences shares the same LCS even if there are extra words in between the consequetive words of LCS, unlike n-grams.

ROUGE is computed using Equations 4.9-4.11, where P_l is LCS precision, R_l is LCS recall, $l(c_i, s_{ij})$ is the length of LCS between the candidate sentence c_i and the reference sentence s_{ij}, $|\,.\,|$ is the absolute value symbol and $\beta = 1.2$.

$$P_l = \max_j \frac{l(c_i, s_{ij})}{|c_i|} \tag{4.9}$$

$$R_l = \max_j \frac{l(c_i, s_{ij})}{|s_{ij}|} \tag{4.10}$$

$$ROUGE(c_i, S_i) = \frac{(1+\beta^2)R_l P_l}{R_l + \beta^2 P_l} \tag{4.11}$$

METEOR [69] is computed by matching the identical words between the candidate and reference sentences. The matching is performed so that the total number of chunks, where a chunk is a series of exact matches, is minimized. METEOR is calculated using Equation 4.12-4.16, where m is the total number of matched words, ch is the number of chunks, P_m is the precision, R_m is the recall, F_{mean} is a parameterized version of harmonic mean of P_m and R_m, P_{en} is the term penalizing the high number of chunks and favoring the exact word matches. α, β and γ take the default values given in METEOR [69] .

$$P_m = \frac{|m|}{\sum_k h_k(c_i)} \tag{4.12}$$

$$R_m = \frac{|m|}{\sum_k h_k(s_{ij})} \tag{4.13}$$

$$F_{mean} = \frac{P_m \, R_m}{\alpha P_m + (1 - \alpha) R_m} \tag{4.14}$$

$$P_{en} = \gamma \Big(\frac{ch}{m} \Big)^{\beta} \tag{4.15}$$

$$METEOR = (1 - P_{en}) F_{mean} \tag{4.16}$$

True positive (TP), false positive (FP), true negative (TN) and false negative (FN) are generally used to evaluate the performance of either an image classification model or an object detection model. Assuming an image from retail dataset contains objects from both class A and class B, we could explain these metrics for class A as follows:

(i) TP → Our model correctly predicted this image to contain an object belonging to class A.

(ii) FP → Our model incorrectly predicted this image to contain an object belonging to class A, whereas there is no such object from class A in this image.

(iii) FN → Our model incorrectly predicted this image not to contain an object belonging to class A, whereas there is such object from class A in this image.

(iv) TN → Our model correctly predicted this image not to contain an object belonging to class A.

Precision is calculated for a class as the ratio of true positives to the sum of true positives and false positives as shown in Equation 4.17.

$$Precision = \frac{TP}{TP + FP} \tag{4.17}$$

Precision is calculated for a class as the ratio of true positives to the sum of true positives and false negatives as shown in Equation 4.18.

$$Recall = \frac{TP}{TP + FN} \tag{4.18}$$

4.3. Experiments

4.3.1. Data Preparation and Optimization

In all experiments, we use *modified ResNet-152* introduced in Section 3.1.1 to extract image information. Following Vinyals *et al.* [3], we use transfer learning to initialize the weights of all but the last two layers of *modified ResNet-152*. The weights of last FC layer is initialized with values drawn from a normal distribution with mean 0 and standard deviation 0.02.

All unique words in the training set is used to construct vocabulary, which results in vocabulary size of 25122. Each token in this vocabulary is represented with a 512 dimensional embedding vector. A LSTM network with input size of 512 and hidden size of 512 is used to process the *modified ResNet-152* output together with the embedding vectors. Then, the output of the LSTM network is passed through a linear layer with output size of 25122 and the softmax function to obtain the probabilities of tokens.

We optimize only the last two layers of the *modified ResNet-152* during the training and use the transferred weights without any change to fit our description generation model to GPU memory. The parameters of embedding matrix, LSTM and linear output layer are also updated.

For the optimization, we use Adam algorithm [70] with learning rate 0.001, $\beta_1 = 0.9$, $\beta_1 = 0.999$ without weight decay. All models are trained for 50 epochs with a batch size of 256, which performed better than smaller batch sizes. The maximum L_2 norm of the gradients is clipped to 0.25 before updating the model parameters.

In order to regularize LSTM network, weight dropping is applied to hidden to hidden weight matrices with probability 0.5. Variational dropout is applied to both embedding vectors and the output of LSTM with probability 0.4. Embedding dropout with probability 0.1, AR with $\alpha = 2$, TAR with $\beta = 1$ and weight tying are also used.

4.3.2. Comparative Analysis

In order to assess the effectiveness of regularization on image captioning, we compare the performance of our base model, which is trained without any regularization method, to its variants, which are trained with such methods. Table 4.1 shows the validation scores for our base model and its variants. The first row gives the performance of our base model, whereas the other rows include the performance of its variants trained with one or more regularization methods.

The inclusion of variational dropout created the biggest jump in CIDEr score, 4.3% increase, and all other scores. Although AR and TAR are also increased CIDEr score, 1.5% increase, they failed to change the other scores on the same level. Other regularization techniques did not make much impact on evaluation scores. However, the addition of all regularization techniques were essential in getting the best scores on most of the evaluation metrics.

Table 4.1. Evaluation scores of model generated captions on the validation set. The first row includes the performance of the base model without any regularization. The other rows shows the performance of the base model trained with one or more regularization techniques. RI column for each metric shows the relative improvement of a model compared to the base model in percentage.

Model	CIDEr		Bleu-1		Bleu-2		Bleu-3		Bleu-4		METEOR		ROUGE-L	
	Score	RI(%)	Score	RI(%)	Score	RI(%)	Score	RI(%)	Score	RI(%)	Score	RI(%)	Score	RI(%)
Base Model	815		679		501		357		254		233		501	
+ weight tying	806	−1.1	681	+0.3	501		356	−0.3	253	−0.4	233		501	
+ weight-dropping	812	−0.4	681	+0.3	501		357		254		233		501	
+ embedding dropout	821	+0.7	688	+1.3	509	+1.6	363	+1.7	258	+1.6	234	+0.4	504	+0.6
+ AR and TAR	828	+1.5	685	+0.9	505	+0.8	359	+0.6	255	+0.4	234	+0.4	504	+0.6
+ variational dropout	**850**	+4.3	694	+2.2	514	+2.6	370	+3.6	265	+4.3	**239**	+2.6	510	+1.8
+ all regularizations	842	+3.3	**701**	+3.2	**524**	+4.6	**376**	+5.3	**267**	+5.1	236	+1.3	**511**	+2.0

4.3.3. Deeper Analysis of Experiments

4.3.3.1. Loss Evaluation Mismatch. We analyze the correlation between the cross entropy and evaluation scores to determine whether our model suffers from loss evaluation mismatch [71, 72]. This phenomenon arises from the fact that we train our model using a word level loss during training phase, whereas we aim to improve sequence level evaluation metrics during inference phase.

Figure 4.4 shows the loss and CIDEr score of our base model with all regularizations after each epoch on the train set and the validation set. There is an inverse relation observed between the loss and CIDEr score. As such, we can state that there is no behavioral mismatch between the metric with respect to which the model is trained and the one with respect to which its performance is evaluated. In other words, our model does not suffer from loss-evaluation mismatch commonly encountered in the literature [71, 72].

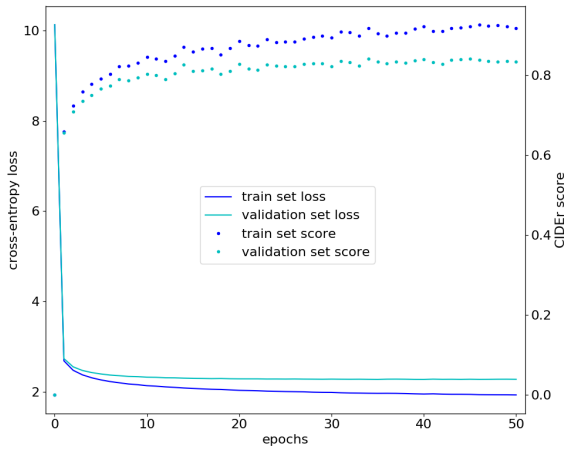

Figure 4.4. Cross-entropy loss and CIDEr score of our base model with all regularizations evaluated after each epoch during the training phase on both the train set and the validation set

We also plot the average probability of a token as a function of its position in a sentence conditioned on the correct tokens up to that position in Figure 4.5 to investigate the effect of token position on the performance of our model. The first word of the sentences in both the training and validation set are predicted with a high probability, whereas we expect it to be very low similar to the results reported in the work of Merity *et al.* [31] on language modeling.

In order to explain this discrepancy, we count the total number of times each word is used as the first word of ground truth sentences in the training set. Then, we take the top 10 most frequent words based on their count and evaluate the percentage of sentences starting with these words. We also count the total number of times each word is used as the first word of model generated sentences in the same set. Then, we take the top 10 most frequent words based on their count and evaluate the percentage of model generated sentences starting with these words. Moreover, we evaluate the same statistics in the validation set. We show these statistics for the training set in Table 4.2 and for the validation set in Table 4.3. The word count is shown in thousands (k) in both tables. Almost 70% of sentences start with "a" word in both sets. Consequently, 98% of sentences generated with our model start with "a" word, reducing the diversity

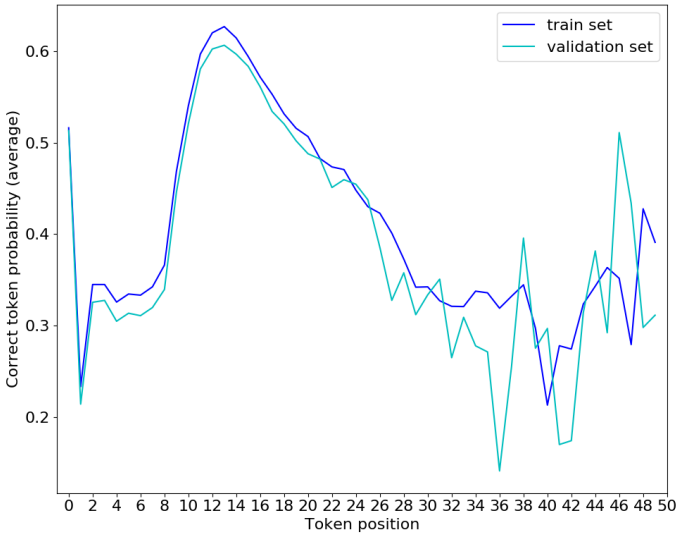

Figure 4.5. Average probability of a token as a function of its position in a sentence conditioned on the correct tokens up to that position

of these sentences. This means that this discrepancy is caused by the memorization of MSCOCO dataset by our model.

Table 4.2. The statistics of the top 10 most frequent first words (both from the ground truth and the predicted sentences) in the training set.

Training Set First Word Statistics					
Ground Truth Words			**Predicted Words**		
Word	**Count(k)**	**Percent**	**Word**	**Count(k)**	**Percent**
a	283.1	68.37	a	405.1	97.83
two	23.4	5.66	two	7.9	1.9
the	18.3	4.42	an	0.6	0.15
an	13.2	3.20	three	0.3	0.07
there	8.4	2.03	four	0.1	0.02
three	5.4	1.30	people	0.05	0.01
people	4.2	1.01	donuts	~ 0	~ 0
several	4.1	0.99	sheep	~ 0	~ 0
some	3.9	0.94	minnesota	~ 0	~ 0
this	3.3	0.80	carrots	~ 0	~ 0

Table 4.3. The statistics of the top 10 most frequent first words (both from the ground truth and the predicted sentences) in the validation set.

Validation Set First Word Statistics					
Ground Truth Words			Predicted Words		
Word	**Count(k)**	**Percent**	**Word**	**Count(k)**	**Percent**
a	68.6	68.57	a	98.190	98.13
two	5.5	5.47	two	1.585	1.58
the	4.5	4.48	an	0.145	0.14
an	3.1	3.12	three	0.100	0.10
there	2.1	2.10	four	0.15	0.01
three	1.2	1.19	bananas	~ 0	~ 0
people	1.0	1.02	donuts	~ 0	~ 0
several	1.0	0.99	yellow	~ 0	~ 0
some	0.9	0.90	pasta	~ 0	~ 0
this	0.8	0.83	orange	~ 0	~ 0

Excluding the first word, the average probability increases up until token position of 15, then shows almost a steady decrease afterwards. We believe that this is related to caption lengths in MSCOCO dataset, for which the total number of tokens at each position is shown in Figure 4.6. As a result, our model is unable to learn long term dependencies.

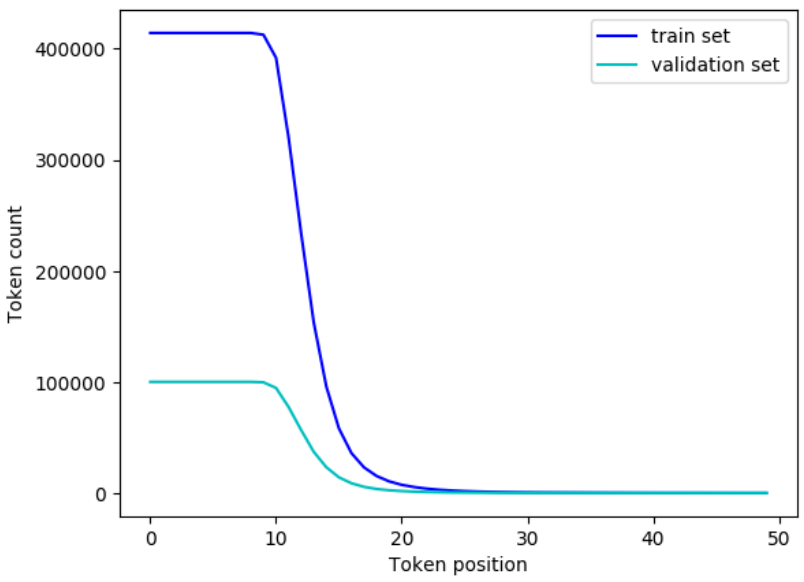

Figure 4.6. Total number of tokens at each position in MSCOCO dataset

4.3.3.2. Word Frequency. In this analysis, we investigate the relation between word frequency and word embedding vector for our image captioning models, similar to the work of Gong *et al.* [73] in language modeling. We evaluate 2-rank approximation of embedding weight matrix from its Singular Value Decomposition using two largest eigenvalues, then plot 2-rank approximation of each embedding vector. Through this section, the top 20% frequent words in the vocabulary is denoted as popular words (blue points in the following figures), while the rest are denoted as rare words (red points in the following figures).

We show 2-rank approximation of each embedding vector of our base model which is trained with all regularization techniques in Figure 4.7. One would expect semantically similar words to be in the close proximity of each other in this 2-dimensional embedding space. Instead of semantic similarity, we observe that the words populate this space based on their frequency. Gong *et al.* [73] suggested that one of the possible reasons for this unexpected behavior is the imbalanced update frequency of embedding vectors during the training phase. Since rare embedding vectors are updated less frequently than popular ones, our model is unable to learn the semantic of a rare word. In order to test this hypothesis, we have measured the frequency of each word in the training set. Figure 4.8 shows the total number of unique words versus word frequency in the training set. We also plot the same graph for only the rare words in the training set in Figure 4.9. We observe that there is imbalanced word distribution in the training set. This observation supports the hypothesis proposed by Gong *et al.* [73].

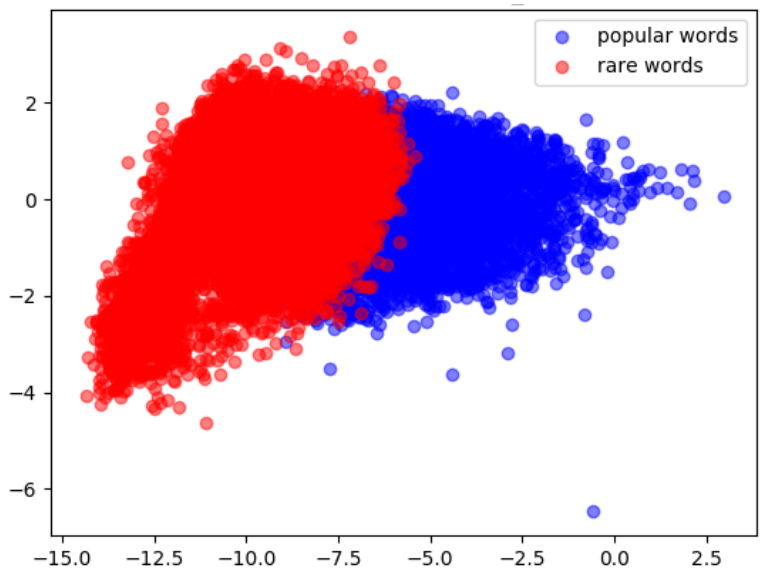

Figure 4.7. 2-rank embedding vector approximation of our base model trained with all regularization techniques

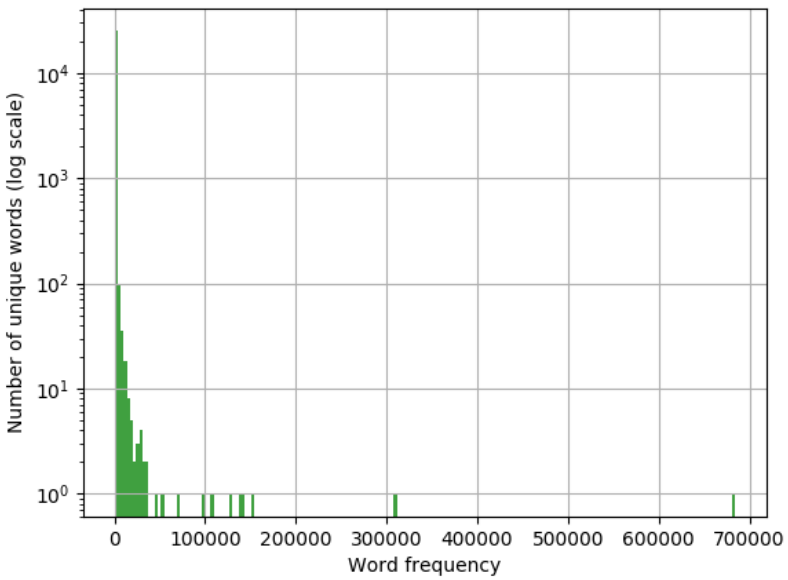

Figure 4.8. Word frequency of all words in the training set

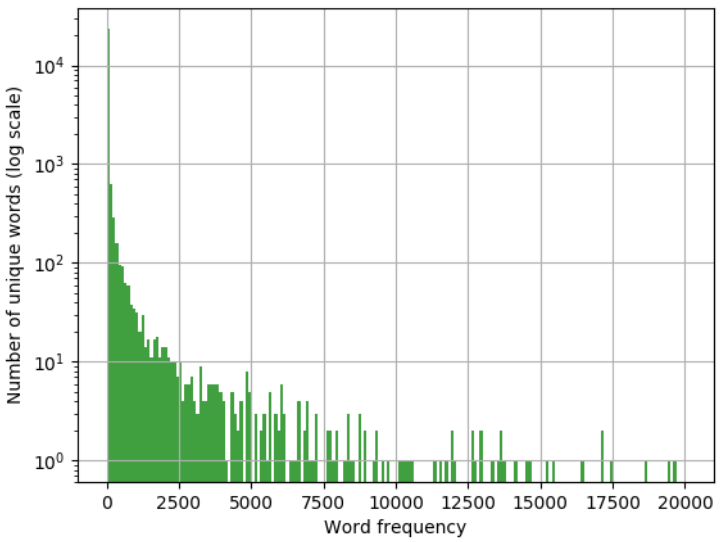

Figure 4.9. Word frequency of less frequent words in the training set

5. APPLICATIONS

5.1. Human in the Loop System

In this section, we introduce a human in the loop description generation system, in which humans provide the start of a caption and the model generates the rest of it. People with speech disorders could use these systems to improve their life standards. Hybrid systems could also be used to shorten the decision making processes for a system where a human interpretation of a visual scene is necessary, such as surveillance. Formally, in order to generate a caption with length of T, we take N consecutive words from the start of a human generated caption and use them to generate the rest of the words. Then, we compute evaluation scores for the generated caption as a function of N.

Figure 5.1. An image from MS COCO dataset

In order to subjectively evaluate human in the loop system, we have listed the sentences produced by our method for an image from MS COCO dataset shown in Figure 5.1 below, where the number before each sentence indicates N (e.g. (5) indicates N=5) and each model generated token is written in italic:

- (0) → *a dog is standing in the grass near a fence . < eos >*
- (1) → black *and white photo of a dog in a field . < eos >*

- (2) → black and *white photo of a dog in a field . < eos >*
- (3) → black and white *photo of a dog in a field . < eos >*
- (4) → black and white photo *of a dog in a field . < eos >*
- (5) → black and white photo of *a dog in a field . < eos >*
- (6) → black and white photo of woman *in a field with a dog . < eos >*
- (7) → black and white photo of woman pushing *a horse . < eos >*
- (8) → black and white photo of woman pushing away *from a dog . < eos >*
- (9) → black and white photo of woman pushing away a *dog . < eos >*
- (10) → black and white photo of woman pushing away a dog *. < eos >*
- (11) → black and white photo of woman pushing away a dog with *a dog . < eos >*
- (12) → black and white photo of woman pushing away a dog with broom *. < eos >*
- (13) → black and white photo of woman pushing away a dog with broom *. < eos >*
- (14) → black and white photo of woman pushing away a dog with broom *. < eos >*

Considering the first sentence, only "black" token is provided by a human and the rest are generated by our model. Notice that same sentences are generated for $1 \leq N \leq 5$ for the same image. This indicates that using only one human generated token improves the performance as much as using five human generated tokens for this particular image. We have also made similar observations for other sentences in MS COCO dataset. In general, our model generated reasonable sentences even for $N = 0$ and the information in the generated sentences have increased slightly as N increases.

Figure 5.2 shows CIDEr score of human in the loop system for our best image captioning model trained with all regularization techniques on our validation set for the objective evaluation of our method. We used the consecutive words from one of the five human generated sentences to generate a caption. Then, we compared this caption with other four human generated sentences to calculate CIDEr score of our system. As expected, validation set score generally increases as N increases. More strikingly, our

results show that providing only a few words was enough to drive up the evaluation scores to levels which are acceptable for practical applications, while reducing human involvement in the process. More concretely, introducing human in the loop system improves CIDEr score of our best model by 30 points using only the first two tokens of a reference sentence of an image.

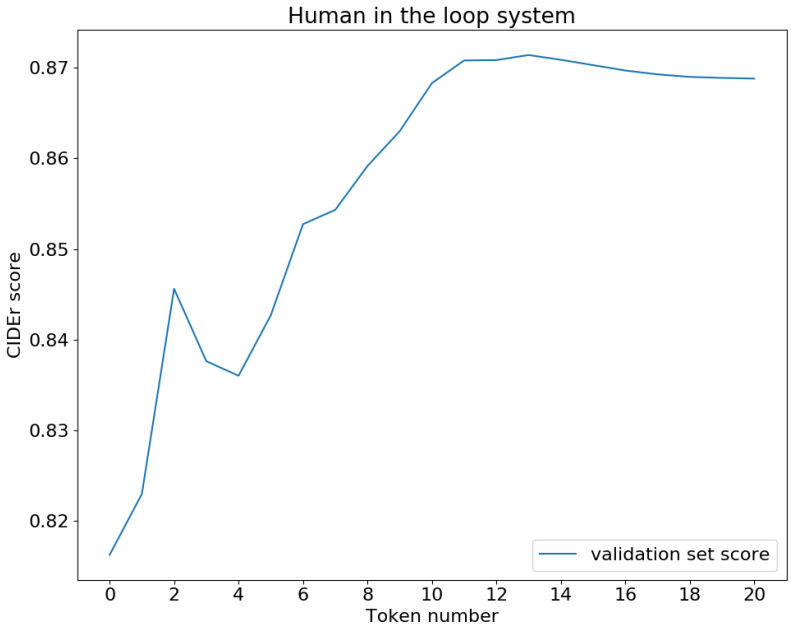

Figure 5.2. CIDEr score of human in the loop system for our best image captioning model trained with all regularization techniques

5.2. Zero Shot Object Detection

Object detection is a computer vision task that aims to detect, localize and classify objects in an image, where each object location is generally annotated with a rectangular bounding box. Neural network based object detection models [23, 74–76] demonstrated high performance on this task over the past few years. Given that these models requires a great number of labeled images and it takes a long time to label each object location, it is desirable to develop a system that only classifies all objects in an image without needing their locations. We train our image captioning model for this purpose and refer to this task as zero shot object detector through this section.

In order to train our model, each image has to be annotated with at least one caption. Assuming that an image is annotated with N labels (e.g. poster, price card and umbrella for $N = 3$), a random permutation of these labels with length N is used as the target caption (e.g. [poster pricecard umbrella], [pricecard umbrella poster] or some other permutation of poster, price card and umbrella) at each epoch during the training phase. We trained our model for 100 epochs on the training set of retail dataset and evaluated its performance on both training and validation set.

Figure 5.3 shows TP, FP, FN, TN, precision and recall scores of our zero shot object detector. We used total tags column to explicitly show the total number of images belonging to each class (i.e. $TP + FN$). As the total number of images per class increases, both precision and recall also increase. In particular, we obtained 0.99 precision and 0.86 recall on the validation set for cabinet class which only has 1080 images in the training set. These results show that our zero shot object detector is a viable alternative to other neural network based object detection models when it is not required to find the locations of detected objects and there are enough images per class in the training set.

	Train Set							Validation Set						
Class	TP	FP	FN	TN	Total Tags	Precision	Recall	TP	FP	FN	TN	Total Tags	Precision	Recall
poster	359	157	26	707	385	.69	385	39	37	11	70	50	.51	.78
umbrella	0	0	4	1245	4	~	4	0	0	0	157	0	~	~
waste bin	0	0	14	1235	14	~	14	0	0	2	155	2	~	0
price card	819	66	63	301	882	.92	.93	86	15	25	31	111	.85	.77
cabinet	1034	2	46	167	1080	.99	.96	116	1	19	21	135	.99	.86

Figure 5.3. TP, FP, FN, TN, precision and recall scores of zero shot object detector

6. CONCLUSION

In this work, we analyzed the effects of a set of regularization techniques on the performance of an image captioning model. We showed that the majority of evaluation scores increases with the addition of all regularizations, while the most significant increase comes from variational dropout. Further, we found that our model does not suffer from loss-evaluation mismatch arising from the difference between word-level training loss and sequence-level evaluation metrics. On the other hand, our model was unable to learn long term dependencies due to lack of long sentences in the training set. Moreover, we showed the effects of imbalanced word frequency on the embedding vectors. Finally, we explored two different applications of our image captioning model, namely a human in the loop image captioning system, in which human intelligence was leveraged to generate more accurate sentences, and zero shot object detection. In the former one, our results showed that incorporating humans in this process increased the performance of our image captioning model significantly even with only a few words, making such systems usable at practical applications. In the latter, we detected each object in an image without finding their locations using our image captioning model.

In the future work, we will explore several directions to improve the performance of our best image captioning model. Firstly, we will use beam search instead of greedy search to generate sentences during the inference phase of our model. Similar to the work of Karpathy and Fei-Fei [1], this could give a little boost to objective evaluation scores. Secondly, we will investigate the effects of pretrained word embeddings on the performance of our model. In the recent years, several different pretrained word embeddings were used to achieve state of art performances on different natural language processing tasks [77,78]. Instead of randomly initializing weights of the embedding layer in our image captioning model, we will use such embedding vectors. Thirdly, we will perform hyper-parameter search for each regularization method. Our objective is to measure the relative importance of each hyper-parameter similar to the work of Merity *et al.* [31] on language modeling. In this way, we could also find the hyper-parameters which result in the highest increase in performance for each method. Moreover, we

will repeat each experiment on several different image captioning datasets such as Flickr8K [79], Flickr30K [80] and Conceptual Captions [81] with several different state of the art image captioning architectures to assess the generality of our results.

We have introduced two different applications for our image captioning model, namely human in the loop system and zero shot object detection. In the former one, we have used consecutive words provided by a human to generate a complete sentence. Given that one would easily detect each object in an image, it is desirable to build a system that could generate a sentence including the classes of these objects. In order to do that, we will explore different ways to provide tokens to our human in the loop description generation system. In the latter one, we have only trained our model on a small dataset, which contains only five different object categories. In order to test the performance of our model, we will train and test our model on ImageNet [4], which contains 200 object categories and thousands of images for each class.

REFERENCES

1. Karpathy, A. and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions", *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3128–3137, 2015.

2. Bernardi, R., R. Cakici, D. Elliott, A. Erdem, E. Erdem, N. Ikizler-Cinbis, F. Keller, A. Muscat and B. Plank, "Automatic description generation from images: A survey of models, datasets, and evaluation measures", *Journal of Artificial Intelligence Research*, Vol. 55, pp. 409–442, 2016.

3. Vinyals, O., A. Toshev, S. Bengio and D. Erhan, "Show and Tell: Lessons Learned from the 2015 MSCOCO Image Captioning Challenge", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 39, No. 4, pp. 652–663, April 2017.

4. Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge", *International Journal of Computer Vision*, Vol. 115, No. 3, pp. 211–252, 2015.

5. Krizhevsky, A., I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", *Advances in neural information processing systems*, pp. 1097–1105, 2012.

6. Lecun, Y., L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, Vol. 86, No. 11, pp. 2278–2324, Nov 1998.

7. Simonyan, K. and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", *arXiv preprint arXiv:1409.1556*, 2014.

8. Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going deeper with convolutions", *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

9. He, K., X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition", *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, June 2016.

10. Mikolov, T., M. Karafiát, L. Burget, J. Černockỳ and S. Khudanpur, "Recurrent neural network based language model", *Eleventh annual conference of the international speech communication association*, 2010.

11. Schuster, M. and K. K. Paliwal, "Bidirectional recurrent neural networks", *IEEE Transactions on Signal Processing*, Vol. 45, No. 11, pp. 2673–2681, Nov 1997.

12. Srikanth, M. and R. Srihari, "Biterm language models for document retrieval", *SIGIR*, Vol. 2, pp. 425–426, 2002.

13. De Marneffe, M.-C., B. MacCartney, C. D. Manning *et al.*, "Generating typed dependency parses from phrase structure parses.", *Lrec*, Vol. 6, pp. 449–454, 2006.

14. Farhadi, A., M. Hejrati, M. A. Sadeghi, P. Young, C. Rashtchian, J. Hockenmaier and D. Forsyth, "Every picture tells a story: Generating sentences from images", *European conference on computer vision*, pp. 15–29, Springer, 2010.

15. Li, S., G. Kulkarni, T. L. Berg, A. C. Berg and Y. Choi, "Composing simple image descriptions using web-scale n-grams", *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, pp. 220–228, Association for Computational Linguistics, 2011.

16. Gupta, A. and P. Mannem, "From image annotation to image description", *International Conference on Neural Information Processing*, pp. 196–204, Springer, 2012.

17. Karpathy, A., A. Joulin and L. F. Fei-Fei, "Deep fragment embeddings for bidirectional image sentence mapping", *Advances in neural information processing systems*, pp. 1889–1897, 2014.

18. Kiros, R., R. Salakhutdinov and R. Zemel, "Multimodal neural language models", *International Conference on Machine Learning*, pp. 595–603, 2014.

19. Brants, T. and A. Franz, "Web 1T 5-gram, ver. 1", *LDC2006T13, Linguistic Data Consortium, Philadelphia*, 2006.

20. Baltrušaitis, T., C. Ahuja and L.-P. Morency, "Multimodal machine learning: A survey and taxonomy", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 41, No. 2, pp. 423–443, 2019.

21. Ngiam, J., A. Khosla, M. Kim, J. Nam, H. Lee and A. Y. Ng, "Multimodal deep learning", *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 689–696, 2011.

22. Frome, A., G. S. Corrado, J. Shlens, S. Bengio, J. Dean, T. Mikolov *et al.*, "Devise: A deep visual-semantic embedding model", *Advances in neural information processing systems*, pp. 2121–2129, 2013.

23. Girshick, R., J. Donahue, T. Darrell and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation", *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

24. Mnih, A. and G. E. Hinton, "A scalable hierarchical distributed language model", *Advances in neural information processing systems*, pp. 1081–1088, 2009.

25. Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge", *International Journal of Computer Vision (IJCV)*, Vol. 115, No. 3, pp. 211–252, 2015.

26. Zhang, W., *State-space search: Algorithms, complexity, extensions, and applications*, Springer Science & Business Media, 1999.

27. Hochreiter, S. and J. Schmidhuber, "Long short-term memory", *Neural computation*, Vol. 9, No. 8, pp. 1735–1780, 1997.

28. Holub, A., P. Perona and M. C. Burl, "Entropy-based active learning for object recognition", *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1–8, IEEE, 2008.

29. Branson, S., C. Wah, F. Schroff, B. Babenko, P. Welinder, P. Perona and S. Belongie, "Visual recognition with humans in the loop", *European Conference on Computer Vision*, pp. 438–451, Springer, 2010.

30. Merity, S., N. S. Keskar and R. Socher, "Regularizing and optimizing LSTM language models", *arXiv preprint arXiv:1708.02182*, 2017.

31. Merity, S., N. S. Keskar and R. Socher, "An Analysis of Neural Language Modeling at Multiple Scales", *arXiv preprint arXiv:1803.08240*, 2018.

32. Alpaydin, E., *Introduction to machine learning*, MIT press, 2009.

33. Karpathy, A., *Connecting Images and Natural Language*, Ph.D. Thesis, Ph. D. Dissertation. Stanford University, 2016.

34. Feller, W., "Law of large numbers for identically distributed variables", *An introduction to probability theory and its applications*, Vol. 2, pp. 231–234, 1971.

35. Goodfellow, I., Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016, `http://www.deeplearningbook.org`.

36. Whittaker, E. T. and G. N. Watson, "Forms of the remainder in Taylor's series", *5.41 in a Course in Modern Analysis*, pp. 95–96, 1990.

37. Erdélyi, A., "Asymptotic expansions of Fourier integrals involving logarithmic singularities", *Journal of the Society for Industrial and Applied Mathematics*, Vol. 4, No. 1, pp. 38–47, 1956.

38. Elizondo, D., "The linear separability problem: some testing methods", *IEEE Transactions on Neural Networks*, Vol. 17, No. 2, pp. 330–344, March 2006.

39. Cover, T. M. and J. A. Thomas, "Elements of information theory 2nd edition", *Willey-Interscience: NJ*, 2006.

40. Boyd, S. and L. Vandenberghe, *Convex optimization*, Cambridge university press, 2004.

41. LeCun, Y., I. Kanter and S. A. Solla, "Second order properties of error surfaces: Learning time and generalization", *Advances in neural information processing systems*, pp. 918–924, 1991.

42. Nair, V. and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines", *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

43. Maas, A. L., A. Y. Hannun and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models", *Proc. icml*, Vol. 30, p. 3, 2013.

44. He, K., X. Zhang, S. Ren and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034, Dec 2015.

45. Clevert, D.-A., T. Unterthiner and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)", *arXiv preprint arXiv:1511.07289*, 2015.

46. Cortes, C. and V. Vapnik, "Support-vector networks", *Machine learning*, Vol. 20,

No. 3, pp. 273–297, 1995.

47. Bengio, Y., P. Simard and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult", *IEEE Transactions on Neural Networks*, Vol. 5, No. 2, pp. 157–166, Mar 1994.

48. Shimodaira, H., "Improving predictive inference under covariate shift by weighting the log-likelihood function", *Journal of statistical planning and inference*, Vol. 90, No. 2, pp. 227–244, 2000.

49. Ioffe, S. and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", *arXiv preprint arXiv:1502.03167*, 2015.

50. Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting", *The Journal of Machine Learning Research*, Vol. 15, No. 1, pp. 1929–1958, 2014.

51. Deng, J., W. Dong, R. Socher, L. J. Li, K. Li and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database", *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, June 2009.

52. Donahue, J., Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition", *International conference on machine learning*, pp. 647–655, 2014.

53. Sharif Razavian, A., H. Azizpour, J. Sullivan and S. Carlsson, "CNN features off-the-shelf: an astounding baseline for recognition", *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 806–813, 2014.

54. Yosinski, J., J. Clune, Y. Bengio and H. Lipson, "How transferable are features in deep neural networks?", *Advances in neural information processing systems*, pp. 3320–3328, 2014.

55. Caruana, R., "Learning many related tasks at the same time with backpropagation", *Advances in neural information processing systems*, pp. 657–664, 1995.

56. Bengio, Y., A. Bergeron, N. Boulanger-Lewandowski, T. Breuel, Y. Chherawala, M. Cisse, D. Erhan, J. Eustache, X. Glorot, X. Muller *et al.*, "Deep learners benefit more from out-of-distribution examples", *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 164–172, 2011.

57. Bengio, Y., "Deep learning of representations for unsupervised and transfer learning", *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pp. 17–36, 2012.

58. Zeiler, M. D. and R. Fergus, "Visualizing and understanding convolutional networks", *European conference on computer vision*, pp. 818–833, Springer, 2014.

59. Wan, L., M. Zeiler, S. Zhang, Y. Le Cun and R. Fergus, "Regularization of neural networks using dropconnect", *International conference on machine learning*, pp. 1058–1066, 2013.

60. Gal, Y. and Z. Ghahramani, "A Theoretically Grounded Application of Dropout in Recurrent Neural Networks", D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon and R. Garnett (Editors), *Advances in Neural Information Processing Systems 29*, pp. 1019–1027, Curran Associates, Inc., 2016.

61. Press, O. and L. Wolf, "Using the output embedding to improve language models", *arXiv preprint arXiv:1608.05859*, 2016.

62. Inan, H., K. Khosravi and R. Socher, "Tying word vectors and word classifiers: A loss framework for language modeling", *arXiv preprint arXiv:1611.01462*, 2016.

63. Merity, S., B. McCann and R. Socher, "Revisiting activation regularization for language rnns", *arXiv preprint arXiv:1708.01009*, 2017.

64. Lin, T.-Y., M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár and C. L. Zitnick, "Microsoft coco: Common objects in context", *European conference on computer vision*, pp. 740–755, Springer, 2014.

65. Chen, X., H. Fang, T.-Y. Lin, R. Vedantam, S. Gupta, P. Dollár and C. L. Zitnick, "Microsoft COCO captions: Data collection and evaluation server", *arXiv preprint arXiv:1504.00325*, 2015.

66. Vedantam, R., C. Lawrence Zitnick and D. Parikh, "Cider: Consensus-based image description evaluation", *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4566–4575, 2015.

67. Papineni, K., S. Roukos, T. Ward and W.-J. Zhu, "BLEU: a method for automatic evaluation of machine translation", *Proceedings of the 40th annual meeting on association for computational linguistics*, pp. 311–318, Association for Computational Linguistics, 2002.

68. Lin, C.-Y., "Rouge: A package for automatic evaluation of summaries", *Text Summarization Branches Out*, 2004.

69. Denkowski, M. and A. Lavie, "Meteor universal: Language specific translation evaluation for any target language", *Proceedings of the ninth workshop on statistical machine translation*, pp. 376–380, 2014.

70. Kingma, D. P. and J. Ba, "Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, 2014.

71. Ranzato, M., S. Chopra, M. Auli and W. Zaremba, "Sequence level training with recurrent neural networks", *arXiv preprint arXiv:1511.06732*, 2015.

72. Wiseman, S. and A. M. Rush, "Sequence-to-sequence learning as beam-search optimization", *arXiv preprint arXiv:1606.02960*, 2016.

73. Gong, C., D. He, X. Tan, T. Qin, L. Wang and T.-Y. Liu, "FRAGE: frequency-agnostic word representation", *Advances in Neural Information Processing Systems*, pp. 1334–1345, 2018.

74. Girshick, R., "Fast r-cnn", *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

75. Ren, S., K. He, R. Girshick and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks", *Advances in neural information processing systems*, pp. 91–99, 2015.

76. Redmon, J. and A. Farhadi, "Yolov3: An incremental improvement", *arXiv preprint arXiv:1804.02767*, 2018.

77. Peters, M. E., M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee and L. Zettlemoyer, "Deep contextualized word representations", *arXiv preprint arXiv:1802.05365*, 2018.

78. Howard, J. and S. Ruder, "Fine-tuned language models for text classification", *arXiv preprint arXiv:1801.06146*, 2018.

79. Hodosh, M., P. Young and J. Hockenmaier, "Framing image description as a ranking task: Data, models and evaluation metrics", *Journal of Artificial Intelligence Research*, Vol. 47, pp. 853–899, 2013.

80. Young, P., A. Lai, M. Hodosh and J. Hockenmaier, "From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions", *Transactions of the Association for Computational Linguistics*, Vol. 2, pp. 67–78, 2014.

81. Sharma, P., N. Ding, S. Goodman and R. Soricut, "Conceptual Captions: A Cleaned, Hypernymed, Image Alt-text Dataset For Automatic Image Captioning", *Proceedings of ACL*, 2018.