MAKING MACHINE CONVERSE BETTER

A Dissertation
Presented to
The Academic Faculty

by

Deepak Goyal   Deepak Agrawal   Nitanshi Mahajan

In Partial Fulfillment
of the Requirements for the Degree
Bachelors Of Technology in
Electronics Department

Indian Institute of Technology (Indian School of Mines) Dhanbad
April 2020

Abstract

The ability to converse proficiently forms one of the most fundamental and natural ways for humans to communicate with each other. Humans can understand each other easily by communicating with one another. However, teaching machines the same can be a cumbersome process and often leads to undesirable results. In this dissertation, we propose several methods by which we can improve human-machine dialogue systems or make machines better at communicating with human beings. We investigate several recent models, architectures, datasets, techniques, etc that can be used to make the modern dialogue systems in production better at conversing with human beings around them. We broadly inspect three ways to make these systems better.

First, we show how these intelligent systems can improve their ability to converse by employing better language models. Better language models have the capability to learn the innate properties, nuances, patterns that lie within a good conversation. Our experiments suggest a better model can produce a much better final reply to the user. We employ several architectures mainly variants of attention architecture, which is a general deep learning architecture to our cause. Our spotlight model is based upon the recently introduced neural transduction model, the transformer model which majorly comprises attention architecture to model between source sentence(s) to target sentence.

Secondly, we investigate how explicit inclusion of tools needed for a conversation like knowledge of the topic being conversed, the persona of the person involved in the conversation affects the overall quality of the dialogue systems. We employ hierarchical neural transduction models to employ knowledge within a conversation system.

Thirdly, we introduce control in our dialogue systems, we model several shortcomings of the conventional neural conversational models like genericness, repetition, specificity, response-relatedness, question-asking, etc. We investigate their variations on our final model's performance.

Together these techniques, models, datasets, etc create a framework for building an intelligent system capable of carrying out a deep consistent conversation with a human being.

Thesis Supervisor: Subrata Bhattacharya
Title: Professor in the Department Of Electronics Engineering, IIT ISM Dhanbad

Acknowledgment

We would like to thank the following people who played a crucial role along different spectrums in our life as undergraduate students and people who were really crucial to this project.

First of all, we would like to thank our project supervisor Prof. Subrata Bhattacharya for providing deep honest comments about our work. The comments always pushed us into bringing the best in ourselves. You were always proactive and honest in replying to queries related to the project.

We are also grateful to Prof. Jitendra Kumar for providing us much needed extension for the completion of the project.

We(Deepak Goyal, Deepak Agrawal) are very much thankful to Rohit Mishra of SILP Lab, IIITA. We are thankful to him for introducing us to the problem of Conversational Agents on which our overall project is based on.

We would also like to thank our friend, Hitesh Kumar for providing us a plethora of ideas related to the project. From finalizing the project to providing us deep psychological thinking on various aspects of the project. His help regarding the installation troubleshoots of certain software was very helpful in producing the results of chapter 4 and chapter 5. Thank you <3.

We are deeply grateful to our parents who provided us with their constant support for every decision we took and for everything. This would not have been possible without their love and support.

We also would like to thank our friends especially Vishnu, Mudassar, Umang, Deepak Jain for their constant support and help.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Overview

In the last decade, there has been quite a development in the field of machine learning particularly deep learning, leading to a surge in artificial intelligence research. Deep learning has taken the center stage in solving some of the complex problems to machines like Image Recognition [1], machine translation [2, 3] etc. Deep learning models have beaten humans in mind-boggling board games like Go [4] which have over 10^{172} combinations. Machine learning has become one of the indispensable technologies that we use every day in almost every aspect of our life. From google mail showing suggestions as to what to write in the mail to autonomous vehicles driving us to our desired locations with absolutely no error at all. Deep learning has taken the world of computer vision by storm as it has surpassed human-level performance in most of their tasks [5–7] Despite the success of deep learning models in computer vision tasks like image recognition, etc. Up until recently, they have struggled to produce the state of the art results on Language processing tasks like machine translation, language modeling, etc. With the advent of transformer-based [3] models, the scenario for the language processing tasks seems to have changed.

Despite the inspiring progress of deep learning into language processing tasks like machine translation, dependency parsing, language modeling, etc brought by transformer and variants based models, the ability to converse consistently and fluently with humans remains an open problem with machines.

Recent advanced personal dialogue systems can be considered as an amalgamation of Goal-Driven/Task-Driven Dialogue Systems and Social or Open-Ended Dialogue Systems.

Goal-Driven/Task-Driven Dialogue Systems

These systems assist or help users to get some task done, they mostly have a goal task to be achieved and are conventionally dependent on the particular domain they are used in. They are used in tasks like making a hotel reservation or booking airline tickets.

Multimodel Conventional task-driven dialogue systems consist of four major components:

Natural Language Understanding Model This model user's utterance into predefined semantic slots.

Dialogue State Tracker It manages the current input along with the context (which is the dialogue history) and outputs the current dialogue state.

Dialogue Policy Tracker It fetches the correct action based on the current dialogue state.

Natural Language Generation Models the output sentence based on the action predicted in the Dialogue Policy.

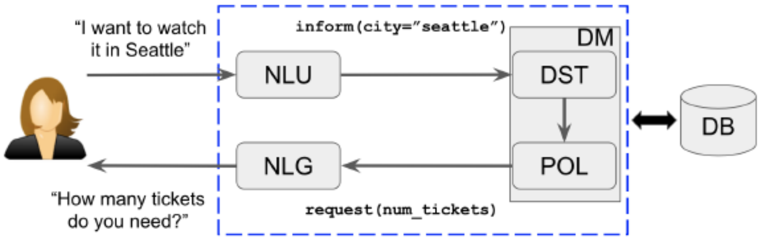

Figure 1.1: Architecture for the task-driven dialogue system. The system as shown consists of four parts: NLU(Natural Language Understanding) Model, DST (Dialogue State Tracker) Model, POL (Dialogue Policy Tracker), NLG(Natural Language Generation Model). The whole system might have access to an external database(DB). Taken from [8].

These components consist of handcrafted models based on machine learning, deep learning and reinforcement learning. Furthermore, each component is dependent on other components which makes it difficult for training the system for some other task. Training it for another task requires too much human effort.

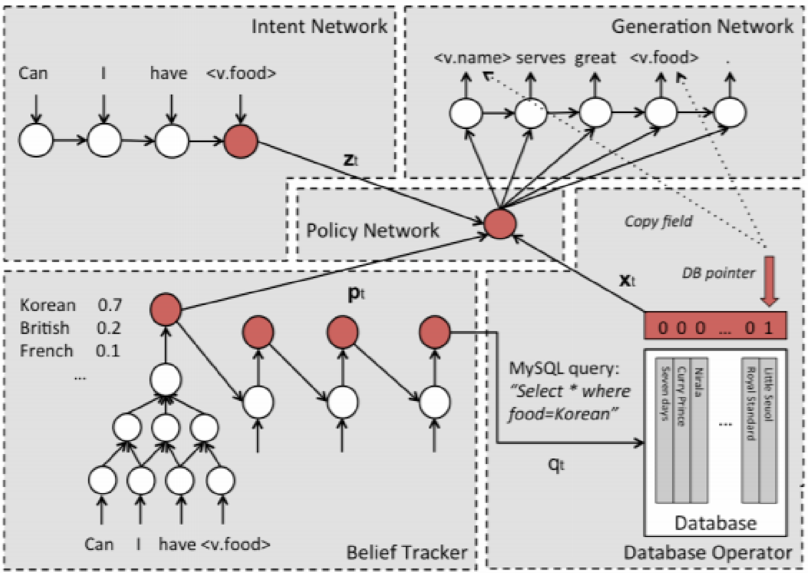

Figure 1.2: An example of an end-to-end task-based generative dialogue system. Taken from [9].

End-to-End Task-Based Dialogue Systems: With the advent of neural generative models mostly after [10], many attempts were made to streamline the task-oriented dialogue systems. End-to-End systems consist of a single module for generating a valid response and action to the user's query. These end-to-end models are usually based on the sequence to sequence mapping [11].

Open-Ended Dialogue Systems

These dialogue systems are more versatile as they answer a wide range of user's queries. Since they are independent of the domain, they are generally referred to as the Open-Ended Dialogue Systems. These models are trained entirely from data without depending on any expert knowledge while designing or augmentation. These open-ended conversational models can converse with humans in a way humans expect other humans to talk to them, therefore, these models are sometimes referred to as the Social Dialogue Systems or Chitchat Dialog Systems.

Open-Ended Dialogue Systems are generally based on neural networks, these systems rely on weights inside the neural networks to learn the innate structure of conversations from the datasets of conversations. These systems nowadays, are generally based on the variant of sequence to sequence architecture [11]. The sequence to sequence architecture consists of an encoder module and a decoder module. The encoder encodes the source sentence to an encoded sentence which is further decoded by the decoder module to target sentence. More on the sequence to sequence architecture in chapter 2.

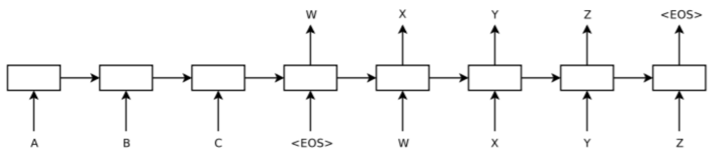

Figure 1.3: Shows the encoder-decoder model proposed by [11]. The model encoded the source sentence and then decodes it using a separate decoder module. The encoder, decoder module can be vanilla RNN, Gated Recurrent Unit(GRU) [12] or a Long Short Term Memory(LSTM) network [Hochrieter and Schmidhuber 1997].

1.2 Making Models Learn the Art of Conversation

One of the many fields that current machine learning/ deep learning researchers are currently working on is to make versatile models that can be employed to learn multiple tasks requiring the least scaffolding when employed on different tasks. As one might expect this has been utterly difficult as we have contrasting innate structure for different types of inputs. A vision model cannot be employed to solve language processing tasks and vice versa. Despite this, many of the language processing models, specifically natural language generation models can be employed in many natural language generation applications requiring very few changes. The most formidable example of this is the sequence to sequence model [11], which was originally released for the machine translation task but is now used in almost every Natural Language Generation task like Dialogue Systems, Question Answering, Language Modelling, etc.

Sequence to Sequence models used to employ Recurrent Neural Networks for the encoder and decoder modules, these Neural Networks suffer from the problem of Vanishing and Explosive Gradient [**y**]. The Vanishing Gradient problem makes it harder for the language models to learn about the long term dependencies and long term structures in the datasets. Explosive Gradient problem causes the updates to take a large value making which can shift our model's configuration to a stage where it incurs large loss value and we have to restart the training process. The solution is to clip the value of gradient if it grows greater than a certain value known as the gradient clipping. More about explosive gradients and the gradient clipping is discussed in the 3rd chapter.

While the problem of the explosive gradient can be solved using gradient clipping, the fact that the Recurrent Neural Network is unable to preserve knowledge over many long ranges. The RNN cannot capture the long term dependencies and structures. The retention of knowledge over long ranges can be solved using a separate memory in our RNN. Hochreiter and Schmidhuber [13] in 1997 presented the Long Short Term Memory [LSTM] model as a solution to the problem of vanishing gradient. Invariably, LSTM architecture makes it easier for the RNN to preserve knowledge and hence long term structures over long ranges, but does not provide a guarantee that there will be no explosive or vanishing gradients.

A simpler alternative to LSTM architecture is Gated Recurrent Unit Architecture [14]. These are also similar to the LSTM architecture in a way that they also help RNN to retain long term info in a similar mechanism. The silver lining for the GRU is that it has lesser parameters than the LSTM architecture and thus is more efficient than them. There is no concrete evidence or study as to which architecture performs better. More about Vanishing Gradient, LSTM, GRU architectures can be found in the 2nd chapter.

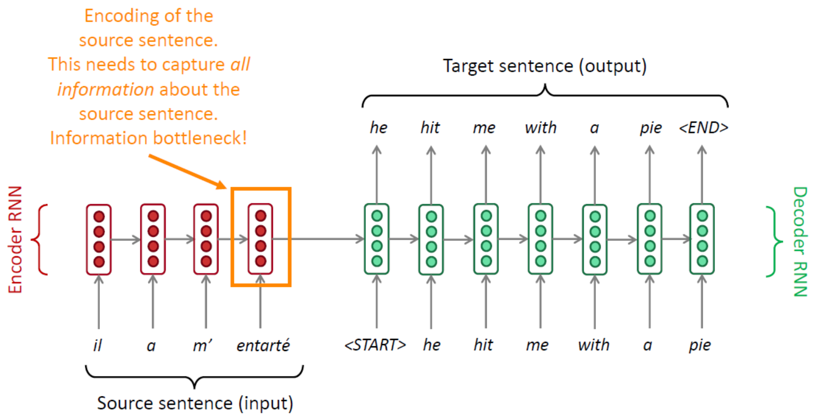

Figure 1.4: The figure shows the bottleneck problem in case of sequence to sequence architecture. Taken from Stanford's CS224N.

The sequence to sequence architecture while being a versatile architecture that can be applied to large language processing tasks like Dialogue, Summarization, Parsing, etc. suffers from the problem of the bottleneck, the last module of the encoder needs to capture all the information of the input sentence(s).

To counter this problem [2] introduced attention mechanisms in sequence to sequence architectures which solve the bottleneck problem as it allows the particular decoder module to look along with each module of the encoder. Attention mechanism also helped the sequence to sequence models with the vanishing gradient problem as it provides bypass routes for the gradients to flow across faraway layers of the network. Attention mechanism also provided a much helpful tool of interpretability in the sequence to sequence networks as now we can look at what part of the source sentence the decoder was focussed on at each timestamp.

All recurrent based sequence to sequence models need to be computed sequentially since the nth hidden layer can only be calculated after calculating (n-1)th layer. Sequential computation inhibits parallelization which in turn leads to slower training.

In 2017 [3] introduced the Transformer model in the seminal paper titled "Attention is all you need", which changed the landscape language processing problems. The top models in almost all of the natural language generation are based on some variant of the Transformer model. Just for inference, the Glue* benchmark consisting of nine language understanding tasks consists of top 20-30 models all based on some variant of the Transformer model.

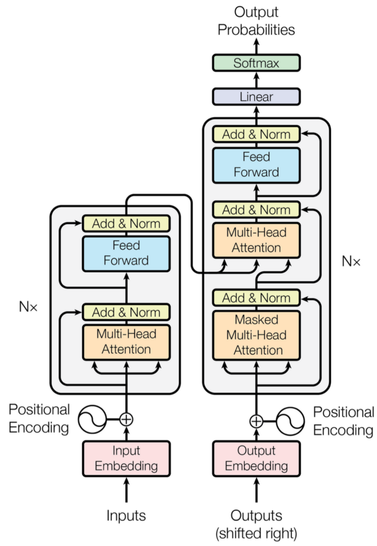

Figure 1.5: The Transformer Model. Taken from [3].

The Transformer model at its core consists of an Encoder and Decoder module similar to sequence to sequence architecture as shown in Fig 1.5. The transformer models allow parallelization unlike the sequence to sequence architecture. The encoder and decoder module consists of a multi-head attention layer which is nothing but a variant of attention mechanism that we introduced before. A detailed description of the model can be found in the 2nd chapter where we formally introduce the Transformer Model.

We have experimented with all the models we have so far described of whose results and inferences can be found in the 3rd Chapter.

1.3 Adding Knowledge, Persona Explicitly

For any good conversation to take place between humans and machines, the machine must have knowledge about the topic of conversation. General knowledge about the world must be known first hand by the machine in order to converse better and answer better user queries.

```
Human: hi
Machine: hi .
Human: how are you
Machine: have you had any before ?
Human: where do you work
Machine: in the house .
Human: where do you live
Machine: you re not a little .
Human: what do you study
Machine: excuse me .
Human: do you have any friends
Machine: i can t do something .
Human: goodbye
Machine: no kiss ?
Human: no
```

Figure 1.6: Some of our early Transformer based neural single turn conversational models. The response of the machine is often out of context. Decoded using Beam Search of beam width 5.

Most of the neural conversational models rely on heavy training on datasets to make knowledge somehow be learned in the weights of their model. These models learn the knowledge embedded in the datasets implicitly. This implicit learning often leads to shallow uninteresting conversations. The conversations are mostly boring and often the response that we get is out of the context of the conversation. Figure 1.6 shows an example of such conversation. We look at how we can model multiple sentences in our model in chapter 4.

1.4 Adding Control

Dialogue Systems or Chitchat models are some of the very open ended tasks in the natural language generation spectrum. The set of correct outputs for a certain input is way too large as compared to any other natural language generation task. This particular notion of having many correct outputs for a single input motivates us to think about parameters that makes a conversation good.

On going through many of these dialogue systems we can say that things that make dialogue systems better include parameters like engaginess of dialogue systems, consistency of the dialogue systems, question-asking, specific responses, etc. For many of us these parameters can be different as per our usage. The other thing that we can look for is to explicitly model these parameters in our model so that our model's response has a certain quantity of these attributes.

In our final chapter, we look at how we can model some of the parameters that we define and how varying these parameters results in a change in the quality of conversation. We examine these changes qualitatively.

1.5 Related Works

The methodology adopted here is based on the rich history of advancements in the fields of linguistics, machine learning, deep learning and application of deep learning to the natural language generation. The dissertation aims to bring these fields together for making dialogue systems that can converse fluently with humans.

Linguistics

Representing images in the computer has been very easy for the researchers as we can just represent easily as a bunch of pixels with certain numeric values mostly in the range 0-255.

Representing words on the computer, on the other hand has been very difficult for Computational Linguistics till today.

People started with representing words as one hot encoded vector of length of the size of the words in a dictionary or the size of vocabulary, which means a certain word is represented as a vector with one at the specified place specifying the word and all zeros(Figure 1.7). The length of the vector is the size of the vocabulary that we have.

$$w^{aardvark} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, w^{a} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, w^{at} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \cdots \; w^{zebra} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

Figure 1.7: The figure shows an example of representing words in a computer using one hot encoding. We represent every word with an R^{V*1} vector with all 0s and one 1 at the index of that word in the sorted english language.

The one-hot encoding vector representation does not have any notion of similarity between words and therefore were unable to capture the popularity and use of researchers. This representation is also sometimes referred to as the denotational semantics.

The idea of similarity between words leads the researchers to use the idea distributional semantics which argues that “The concept of representing the meaning of a word based on the context in which it usually appears”. It is dense and better captures the similarity between words. The current word vector representation models word2vec, glove,

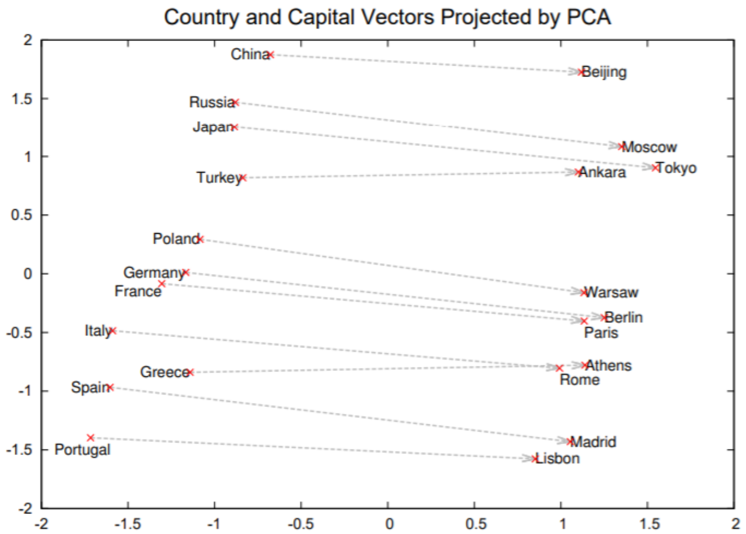

Figure 1.8: Two dimensional representation of words using Word2Vec model. As one can infer, the notion of similarity is quite clear. Taken from [15].

elmo [15–18] are based on attending to context to generate a dense vector representation for words in the computer. Word2Vec used a neural network to learn representation of words, while glove employed both statistical and neural methods in forming the representation for the words.

The success of these dense representations of word using vectors leads to state of art systems like Spell Checker, Keyword Searching, Finding Synonyms, Machine Translation, etc.

Deep Learning and its Application to Natural Language Generation

Deep learning has been quite successful as we have seen in case of representing words, it has also been very much successful in Natural Language Generation tasks like Machine Translation, Language Modelling etc. The advent of neural machine translation [11] opened the gates for solving tasks like language modelling, dialogue systems, etc through neural networks.

While [11] introduced sequence to sequence models based on recurrent neural networks for the machine translation task, [**vinyals**] was the first one to show the use of sequence to sequence architecture for dialogue task. From then, there has been a large body of work in this direction. Citing all of them will take another such dissertation meant for this purpose only. However, we have cited some major of them that we take inspiration from.

[2] introduced the mechanism of attention applied to sequence to sequence architectures for machine translation. The attention mechanism helps us with the bottleneck problem and provides relief from the vanishing gradient problem. Attention mechanism also helps us in providing much needed interpretability in these neural methods.

[3] introduced a new neural transduction model applied to machine translation. This model is the reason behind all the success of the various problems today in Natural Language Processing. The model was so successful that it is now applied to other problems like Images [19], music generation [20] etc and have been cited 7222 times. We have also made use of the Transformer model in our project.

[21] showed how multiple sentences can be encoded using the Hierarchical Encoder-Decoder model. The model was useful in including multiple sentences of context along with the present response.

[22, 23] showed how we can include personal knowledge explicitly in our model. [23] included a separate knowledge base for each turn of dialogue. [22] introduced how to include personas in the dialogue setting. Persona Chat [22] dataset was also the part of ConvAi competition. FAIR group has also combined such tasks in a single software framework called ParlAI which is used as a de-facto research platform for dialogue research.

Controlling the various aspects of the conversation forms one of the major tools for us to make the conversation ability of machines better. To accomplish this, several efforts have been made. These efforts include both modifications during training and during testing as well. Conditional Training efforts include the likes of [24–26]. Modifications during testing or decoding includes [27, 28].

Finally [29] which combines these efforts have been a major inspiration for this thesis.

1.6 Contributions and Achievements

The aim of this dissertation as can be evident from the title itself is to make machines/computers converse better with humans in an open domain setting. The dissertation also sets out to investigate models available in theory to practice in a particular field of open domain conversation models. With that, the dissertation accomplishes the following:

- The dissertation acts as a kind of survey of dialogue systems based on neural networks recently developed across academia and industries.
- Acknowledging the problems related to the single turn dialogue systems.
- Acknowledging the problems related to dialogue systems without explicit modelling of control, knowledge etc.
- Implementation of the several of the recent models made for dialogue systems.
- Acknowledgement of the idea that including attributes like knowledge, controllable parameters leads the way for making a holistic dialogue agent.

The project leads to the formation of library of ready to use implementation of various neural models which can be found here: Conversational-Agents. Further progress in the field of open-ended conversational models will require in-depth understanding of Deep Learning, Deep Learning Based NLG models, machine learning, Linguistics, etc.

Methods | 2

2.1 Overview

Machine learning is a part of Artificial Intelligence that involves computer algorithms that can learn from the data without being explicitly programmed or hardcoding. In this part "learn" means from Mitchell(1997) "A computer program is said to learn from experience E with respect to some class of tasks T and performance measures P, if its performance at tasks in T, as measured by P, improves with experience E'. In practice, machine learning means to find patterns over large data. These patterns lead us to develop solutions for problems that involve a fixed number of mathematically defined constraints that can be programmed into the computers. For example Given certain data points trying to find the best curve that fits them all. There has been a drastic surge in the success and development of machine learning in recent years. Deep Learning takes the credit for the most of it.

This dissertation extensively makes use of methods from deep learning which is a specific kind of machine learning. In order to understand the deep learning, one must know and understand machine learning. Section 2.1 introduces briefly to the concepts of machine learning. Section 2.2 briefly introduces the concepts related to deep learning and computational linguistics. The aim is not to give a comprehensive review of each and every concept but to provide enough for the reader to easily comprehend what lies ahead in the thesis. For more in-depth review and explanations please see the following - [30] for Machine Learning, [31, 32] for Deep Learning.

2.2 Machine Learning

Machine learning involves extracting patterns from the data. The data which we feed into our machine learning system usually is represented as a tensor(an array with more than two axis) of dimension (M*(N1*N2,..)) where M is the number of independent data points in the dataset and N1*N2,.. represents the dimension of each data point corresponding to that particular dataset. Each point of information included in the representation of the datapoint is referred to as the feature.

Problems

Machine learning problems can be broadly classified as supervised problems, unsupervised problems based on the datasets they are provided during the time of training or learning.

A large chunk of the machine learning problems revolves around supervised learning. Supervised learning algorithms or systems come across datasets consisting of data points with corresponding labels or targets. For example, the image recognition task usually involves a dataset consisting of images with their corresponding annotations or class to which a particular image belongs. A supervised learning algorithm can be trained on that dataset to correctly classify a new image into the predefined classes based on the feature of that new image.

Unsupervised learning algorithms come across datasets containing many features, then learn useful properties of the structure of this dataset. An example of unsupervised learning is clustering, which consists of dividing the dataset into clusters of similar examples based on their features.

We will look in detail about the supervised learning in the following section and will have a motivating example of unsupervised learning problem after that.

All problems that we might solve with machine learning or deep learning do not fall clearly into the category of supervised/unsupervised category. There is no formal definition of supervised/unsupervised learning. We describe some of the problems in this thesis which can be easily structured into one of the categories. For more problems and use cases please see [31].

Supervised Learning

In this section, we will dive into supervised learning as a use case to showcase some important machine learning concepts, like what a loss function is and what optimising a loss function w.r.t some variables means.

In supervised learning, we are trying to learn to predict the label or target y for a certain datapoint x. To learn this mapping from a datapoint x to a label y, we need to learn a mathematical function f which takes in the input datapoint x and correctly outputs the true y associated with the datapoint x. To make our function/mapping better at predictions, we need to optimise a "loss function" with respect to the parameters of our function or mapping. The loss function should be such that it should be able to tell when our function is predicting bad results and when it is performing good. The loss function should output a large value when the predicted value by our function is far from the true or actual value and should output a low value when the predicted value is near to the actual or true value corresponding to a certain datapoint. The loss function is a direct metric for the performance of any machine learning system.

Consider a linear regression problem as the problem to build a system that can take a vector x (belongs to sign) R^n and outputs a scalart y (belongs to) R. Let $\hat{y}$ be the value predicted by our function/model and y to be the actual value that it should have predicted. For correct output $y = \hat{y}$.

$\hat{y} = W^t * x = (w_1 * x_1 + w_2 * x_2 + ...w_n * x_n)$

The $W \in R^n$ is the parameter vector.

Parameters are the values that control our function to map from x to $\hat{y}$. We can think of them as the weights that determine how each feature of our datapoint is going to affect the prediction. If a feature xi receives a positive weight wi then increasing the value of that feature increases the value of our prediction $\hat{y}$. If a feature receives a negative weight wi, then increasing the value of that feature decreases the value of our prediction $\hat{y}$. If a feature has large weight than that feature has a larger effect on the prediction than the lower corresponding weight.

One way of measuring the performance of our function is to compute the mean squared error on the test dataset. Let $\hat{y}^{test}$ be the predicted value on the test datapoint and y^{test} be the true or actual value on the test datapoint. Then mean squared error is defined as:

$$MSE_{test} = \frac{1}{m}\sum_i(\hat{y}^{test} - y^{test})_i^2$$

To make our function better at predicting the value y we need to improve on our weights w in a way that reduces MSE_{test} while learning through the training set (X^{train}, Y^{train}). One way to do that is by just minimising the training set or MSE_{train}.

To minimise MSE_{train}, we can clearly solve for where its gradient with respect to the parameters is zero.

$$\Delta w MSE_{train} = 0$$

Whose solution is given by:

$$w = (X^{(train)T}X^{(train)})^{-1}X^{(train)}y^{(train)}$$

For full proof of this please see [31] section 5.1.4.

The system of equations whose solution is given by the above equation are referred to as the normal equations.

A bias term is also introduced in our original function to make the predictions biased towards a certain point in case of absence of inputs or all zero inputs. The modified function looks:

$$\hat{y} = w^T x + b$$

Where $b \in R$.

We can modify our above function for binary classification by making use of logistic sigmoid function(σ)(figure 2.1).

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

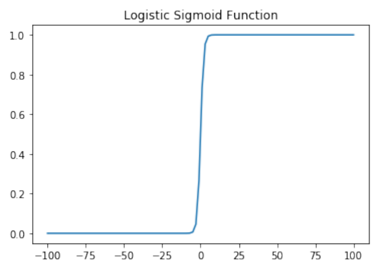

Figure 2.1: The logistic sigmoid function.

The sigmoid gives output in the range between 0 and 1. This makes our function learn the probability that a certain datapoint x belongs or does not belong to a certain class.

$P(y = 1|x) = \sigma(w^T x)$ or $P(y = 0|x) = \sigma(w^T x)$ We can model out function in any way depending on how we model our loss function. To make a decision as to which class the x belongs to we can set a threshold value. If the value of P is greater than 0.5 than the datapoint belongs to class 1 or class 0. If the value of P is less than 0.5 than the datapoint belongs to class 0 or class 1.

Classification models learn a decision boundary that separates the data points on input space or Rn space into the regions where the model predicts one class versus the other. Figure 2.2 shows such an example of decision boundary. There are other classification algorithms like Support Vector Machine which, given data points, finds the maximum margin separating the classes.

For more in depth review of above methods and other machine learning techniques please look at [31] and [30] .

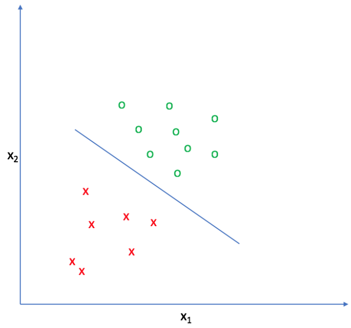

Figure 2.2: Decision boundary separating two classes of data points using Logistic Regression. Taken from Jeremy Jordan 2017.

Generalisation, Overfitting, & Underfitting

The central challenge in machine learning is that our model must perform well on test data that is the data which is not seen by the model during its course of the training. The ability to perform well on the previously unseen data by the model is referred to as the generalisation.

In our above example of linear regression we trained our model by minimising training loss function by optimising parameters of our function or model. The training error is given by

$$\frac{1}{m_{(train)}} \sum(\hat{y}^{(train)} - y^{(train)})^2$$

But in reality we actually care about the test error or loss given by

$$\frac{1}{m_{(test)}} \sum(\hat{y}^{(test)} - y^{(test)})^2$$

The factors determining how well a certain machine learning model will perform depends on two important factors: 1. Training Error (Should be small) 2. Gap between training error and testing error(Should be small).

These two factors correspond to two important challenges in machine learning underfitting and overfitting. Underfitting occurs when the model is unable to make training error/loss small. Overfitting occurs when the gap between training error/loss and testing error/loss is large.

The control over a model's ability to overfit or underfit can be controlled by something called capacity of the model. Capacity of a model is its ability to fit a wide variety of functions. Models with low capacity may struggle to fit the training data thus underfits the model. Models with high capacity may learn the structures of training data in its weights and overfits on the training data.

Machine learning systems work best when their capacity is optimal with the true complexity of the problem at hand. Models with insufficient capacity are unable to solve complex tasks. Models with high capacity can solve complex tasks but models with capacity higher than required are susceptible to overfit the data. Fig 2.3 shows such an example. The required polynomial degree for perfect fit of the data points is two but with a linear polynomial it underfits and with polynomial of degree 9 it overfits.

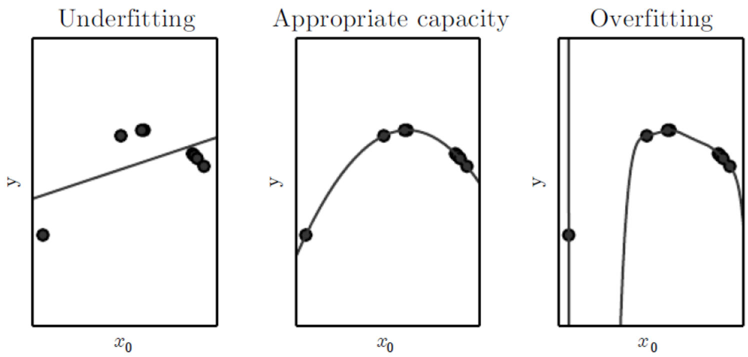

Figure 2.3: First figure shows underfitting on using a linear polynomial. Second figure shows the right fit with the correct quadratic polynomial. Third shows overfitting by a 9-degree polynomial. The third is able to rightly fit the given points but it is also able to fit many other data points with these points. Taken from [31].

Regularisation

Regularisation is any modification we make to a learning algorithm to make it generalised. It is not concerned with training loss.

Hyperparameters

Settings that are used to control the behaviour of machine learning algorithms. These settings are hyperparameters. The values of the hyperparameters are learned on their own by the learning algorithms and are usually assigned based on experience.

2.3 Deep Learning

Deep learning is the subset of machine learning that generally deals with a large number of linear layers stacked together. Deep learning is also sometimes referred to as representational learning along with hierarchical learning. This is due to the fact that layers in a deep learning model learns the optimal representation of the raw data that we have to rightly solve our problem which can be tasks like image recognition, etc. The representations of far layers depends on the representations of previous layers, which makes for the hierarchical learning.

Deep learning involves training different kinds of artificial neural networks. A simple neural network consists of hidden layers, an input layer or visible layer and an output layer. Figure 2.5 shows an example of a neural network with two hidden states.

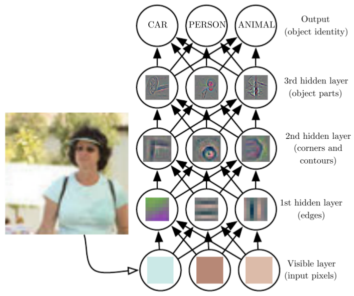

Figure 2.4: Learning a mapping function from an input like an image's pixel value to an identity is an insurmountable difficult task for a machine. Deep Learning helps us make this complex mapping simple by dividing it into a set of simple mappings layer by layer. In the figure above the 1st hidden layer learns about corners and edges, the second one learns about contours and so on. Taken from [33] through [31]

The first layer Input Layer or the Visible Layer: $X \in R^n$ This is the input to the model. In above example $n = 92$.

First Hidden Layer It is computed from the input layer as:
$H(1) = Activation(Wx); H(1) \in R^m$. In above example $m = 300$. W is the parameter and is of dimension $(m * m)$ hence $W \in R^{m*n}$.

Second Hidden Layer $H(2) = Activation(W * H(1))$; $H(2) \in R^m$. In above example $m = 12$. $W \in R^{12*300}$.

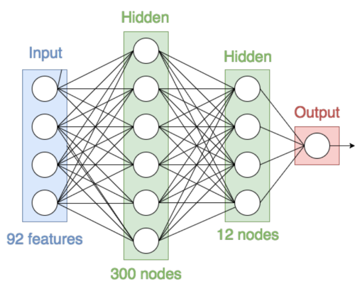

Figure 2.5: A simple neural network with two hidden layers.

Output Layer $H(3) = Activation(W * H(2)); H(3) \in R^1. W \in R^{12*1}$.

Activation Function are functions like sigmoid function we introduced previously, other activation functions include tanh, ReLu(Rectified Linear Unit) etc. Introducing activation functions in between layers introduces non-linearity transformation which helps in learning complex mapping. Applying activation function to the output of the linear transformation yields non-linear transformation or in some cases piecewise linear.

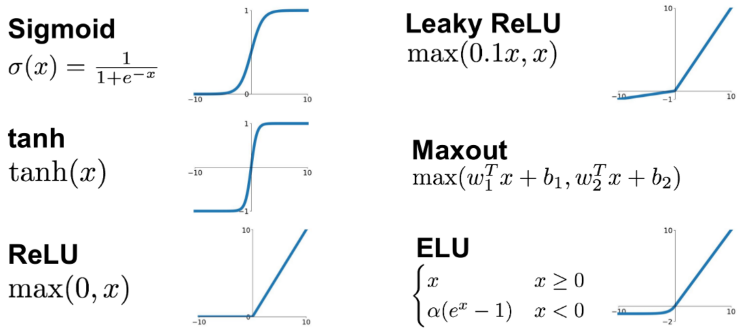

Figure 2.6: Some commonly used activation functions used in Deep Learning.

The Rectified Linear Unit (ReLu) is mostly used due to the fact they generate nearly linear output which helps us in learning with gradient based methods. They also preserve many properties which make linear models to generalise well. We can build a universal function approximator from rectified linear functions. Gradient-Based Learning

As we have seen in our previous section, training a machine learning algorithm involves minimising a loss function with respect to the parameters of our model. We can visualise this problem as having different loss function values for different parameters values, we are interested in finding those values of parameters which correspond to the minimum value of the loss function. Traditional machine learning algorithms used to have a smooth convex loss surface, these loss functions are easy to optimise. However, due to a number of linear layers and non-linear activation layers in between neural networks have very complicated non-convex loss surfaces which makes them harder to optimise. Due to this, we can't use linear equations described in the previous section to optimise them.

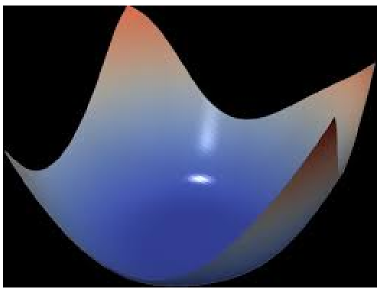

Figure 2.7: The convex loss surface. The blue indicates a low loss and red indicates a large value of loss.Taken from [34].

To find parameters, that minimises the loss function, deep learning adheres to iterative gradient-based optimisation. The optimal parameters minimising the loss is found by descending in the loss landscape starting from a random initialisation of parameters.

The simplest example of such an algorithm is the stochastic gradient descent (SGD). At each step of SGD, we take a training sample randomly from the training set. We then input the training sample into our model to get prediction, using prediction and the true output we compute the loss. We then compute the gradient of the loss with respect to the parameters of the models, and move the parameters by a small amount in the direction of the gradient. The amount of the movement can be controlled by us using a learning parameter (alpha). The learning parameter is one of the hyperparameters of the model.

Algorithm 1: Stochastic Gradient Descent (SGD)

Require: α (learning rate)
Require: w_0 (initial parameters)
$t \leftarrow 0$ (intialize timestep)
while w_t not converged **do**
$X_B \sim X_{train}$
$g_t \leftarrow \nabla_w L(\hat{X}; w)$
$w_{t+1} \leftarrow w_t - \alpha g_t$
$t \leftarrow t + 1$
end

The gradient with respect to the parameters can be calculated using effective backpropagation algorithm [35] i.e by applying recursive chain rule of differentiation. Deep learning frameworks such as pytorch use automatic differentiation using backpropagation and computation graphs [36].

The order of the complexity of the stochastic gradient descent algorithm is linear in size of the batch of examples sampled from the training data. Low batch size is more efficient and requires less memory to run but introduces noise to gradient updates[Dinh 2018]. Most modern alternatives to SGD often rely on computing a moving average of gradient estimates as a form of momentum that affects gradient updates [37]. Others use an adaptive learning rate for estimating the size of updates made at each step such as Adaptive Momentum Estimation(AdaM) [38].

Neural Network Architectures

We have so far discussed the fully connected layers in which one neuron in one layer is connected to every other neuron in the previous layer. This makes it to learn a unique set of parameters for every input feature. This motivates us to consider architectures that have same parameters across layers so that we can exploit similar regularity across the dataset. One such example includes that of Convolutional Neural Networks(CNN) **[le]**. These networks contain learned filters which are applied across in between the layers. The inputs in these networks are mostly the images. Convolutional Neural Networks is an example of weight sharing across the space.

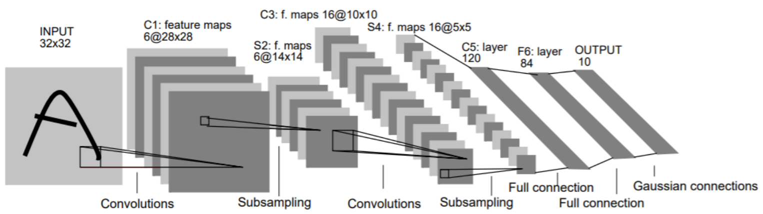

Figure 2.8: The architecture of Le Net for digit recognition. Taken from [39].

The weights of the networks can be shared across time as well. Recurrent neural network is one such example. These networks works on output received from the previous cell(Figure 2.9).

This architecture can also lead to explosive or vanishing gradient as the weights are repeatedly multiplied. If the parameters are (>1) then

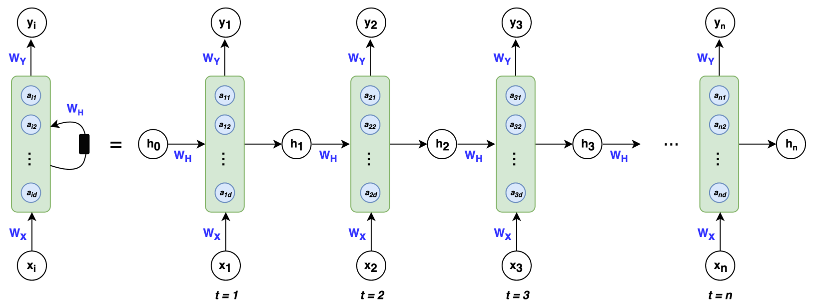

Figure 2.9: The Recurrent Neural Network Architecture.

we can have an exploding gradient or if parameters are (<1) then we can have a vanishing gradient. [Schmidhuber 1998] proposed a solution to this problem through an architecture called Long Short Term Memory networks. The network consists of a forget gate, input gate and output gate which controls the flow of information from one cell to another. Due to these gates we have a kind of highway-bypass connection which helps us with vanishing gradient problems. [14] introduced a similar architecture called Gated Recurrent Unit which is more effective than the LSTM network.

Recently a transformer[3] has been proposed as an alternative to the recurrent neural networks. Transformers uses attention mechanisms to nicely focus on parts of the context for various tasks like text generation or language modelling [40].

2.4 Conclusion

The chapter has briefly captured some of the core concepts in machine learning and deep learning. While we have provided a glimpse of information required to read and understand this thesis, there are more practical and theoretical concepts within the above topics that we could not cover here. For more theoretical and practical review for these concepts please read the following books. [30] for theoretical concepts in machine learning. [31] for theoretical deep learning. [32] for understanding practical deep learning.

Model Architecture Changes 3

3.1 Overview

Natural Language Generation tasks such as machine translation, sentence compression, abstractive summarization, story generation, Dialogue generation, etc all require a sentence transduction model. A general sentence transduction model takes in a group of the sentence(s) and transforms them into another group of the sentence(s). As with any other machine learning problem to learn the structure of mapping from the source sentence(s) to the target sentence(s), we require Sentence Transduction Models to learn how to correct output sentence(s) given a new set of input sentence(s). These models rely on different types of neural network architectures for their functioning. Mostly encoder-decoder type architecture is used with attention mechanism [2, 41] or attention-based architecture like Transformer [3] is used.

This chapter provides an in-depth study of two of the two major Neural Conversation Models that we use today namely Encoder-Decoder based on RNN with Attention and Encoder-Decoder based on Transformer. We first discuss the theoretical aspects of these architectures, then we dive deep into how these models perform on open-ended dialogue task. We also discuss the efficiency of these models with variation in the input and architectural changes. Various decoding algorithms have also been discussed in great detail.

3.2 Neural Sentence Transduction Models

Almost every sentence transduction model is based on the Sequence To Sequence architecture [11]. The architecture consists of two parts- An Encoder and a Decoder. The idea behind the model is to encode a source sentence(s) into a vector representing the source sentence. This transformed representation then is used to generate the target sentence(s) via an encoder like model called the decoder(Figure 3.1)

The encoder and decoder modules in the sequence to sequence architecture are neural models. We generally have two types of neural models in place of encoder and decoder modules in sequence to sequence architecture. These are

1. Recurrent Neural Network(LSTM, GRU) based Encoder-Decoder.
2. Transformer based Encoder-Decoder.

We will now sequentially discuss each one of these.

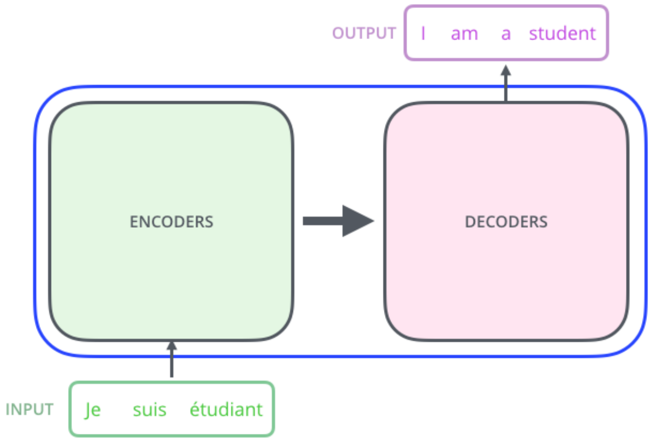

Figure 3.1: Sequence to sequence model consists of two subparts encoder and decoder. Decoder acts as the language model conditioned on the encoder's output, hence problems targeted by the sequence to sequence architecture are also referred to as the conditional language modeling tasks. Taken from Jalammar's blog. Language Modeling is the task for predicting the next word(s) given the context.

Recurrent Neural Network(LSTM, GRU) based Encoder-Decoder

Recurrent Neural Networks are generally used to model sequences like sentences, time series, etc. Recurrent Neural Networks are the neural networks that share their weights along the time, by this, we mean that these networks have similar weights across any two tokens of the sequence. This helps us in capturing the structures and properties of sequences.

The words in the input sentence or the source sentence are first converted into one-hot encoding then this one-hot encoding of each word is converted into their corresponding embedding using an embedding lookup table. These embeddings are then used to calculate the sequential hidden states of the RNN(Figure 3.2).

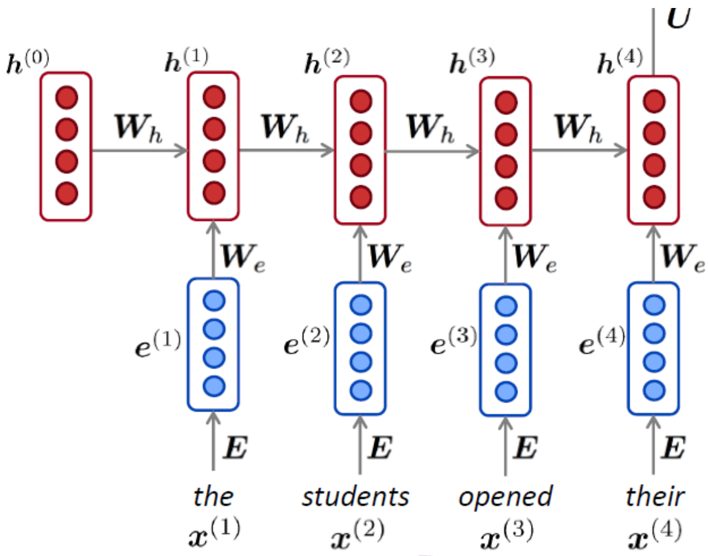

Figure 3.2: The x represents the one-hot encoding of words appearing across the sentence. E represents the Embedding matrix which converts the one-hot encodings into embedding vector representing that particular word. Taken from Stanford's CS224N course.

Mathematically,

$x(k) \in R^V$, where V is the vocabulary size.

$e(k) = Ex(k)$, $E \in R^{(d\_model*V)}$, d_model is the required embedding size

Hidden states are calculated as:

$h(k) = \sigma(W_h(h(k-1)) + W_e(e(k)) + b1)$

The initial hidden state of the encoder is generally an all-zero vector. The final hidden state of the sequence is then passed onto the decoder in case of sequence to sequence architecture. The decoder is also based on this similar mechanism with the only difference that the initial hidden state of the decoder is the encoder's final hidden state.

The decoder also needs an input just like the encoder, this differs in case of training and testing of the model. This type of RNN is also called as the Vanilla RNN.

During Training

We start by giving decoder an input such as <START> token, the first hidden state is calculated using this token's embedding and encoder's last hidden state as the input. The hidden state is passed through a linear to predict the next word probabilities. For the second hidden state, the target's first word along with the first hidden state is used for the computation(Figure 3.3). This process is reported up to some predefined length or until the decoder's outputs <END> token. This process of forcing the target variables into the decoder is sometimes referred to as the teacher forcing.

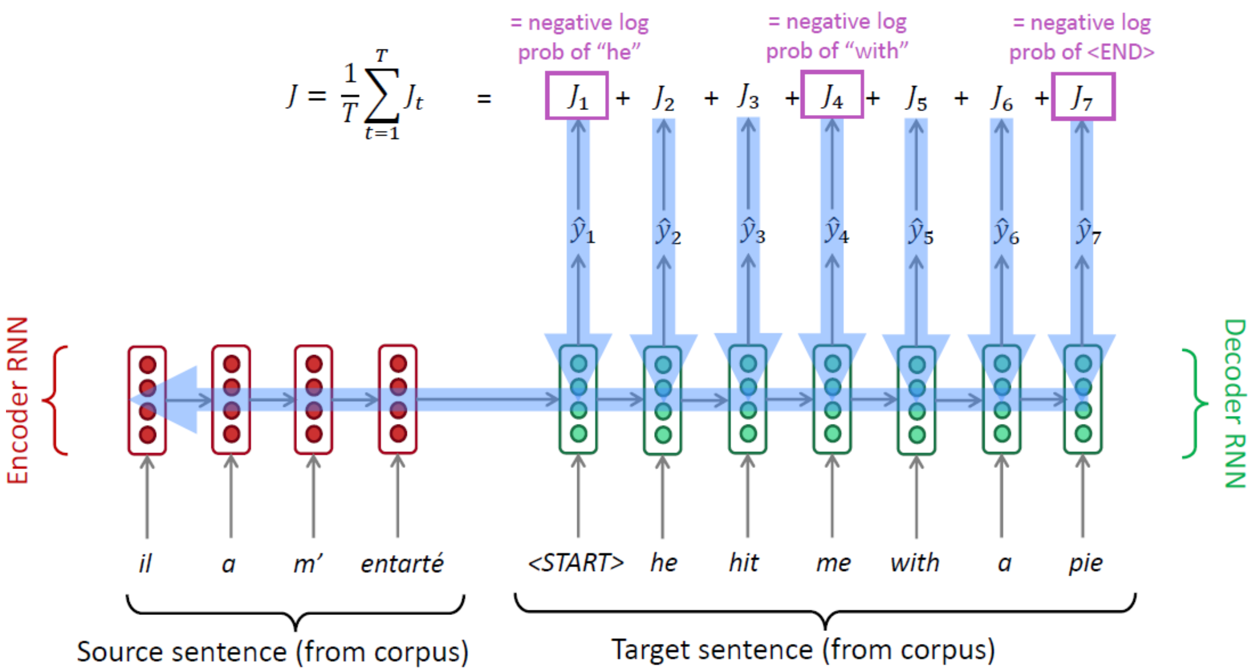

Figure 3.3: The decoder's inputs during training.Taken from Stanford's CS224N.

During Testing

The output predicted at each decoder step is fed into the input of the next step(Figure 3.4).

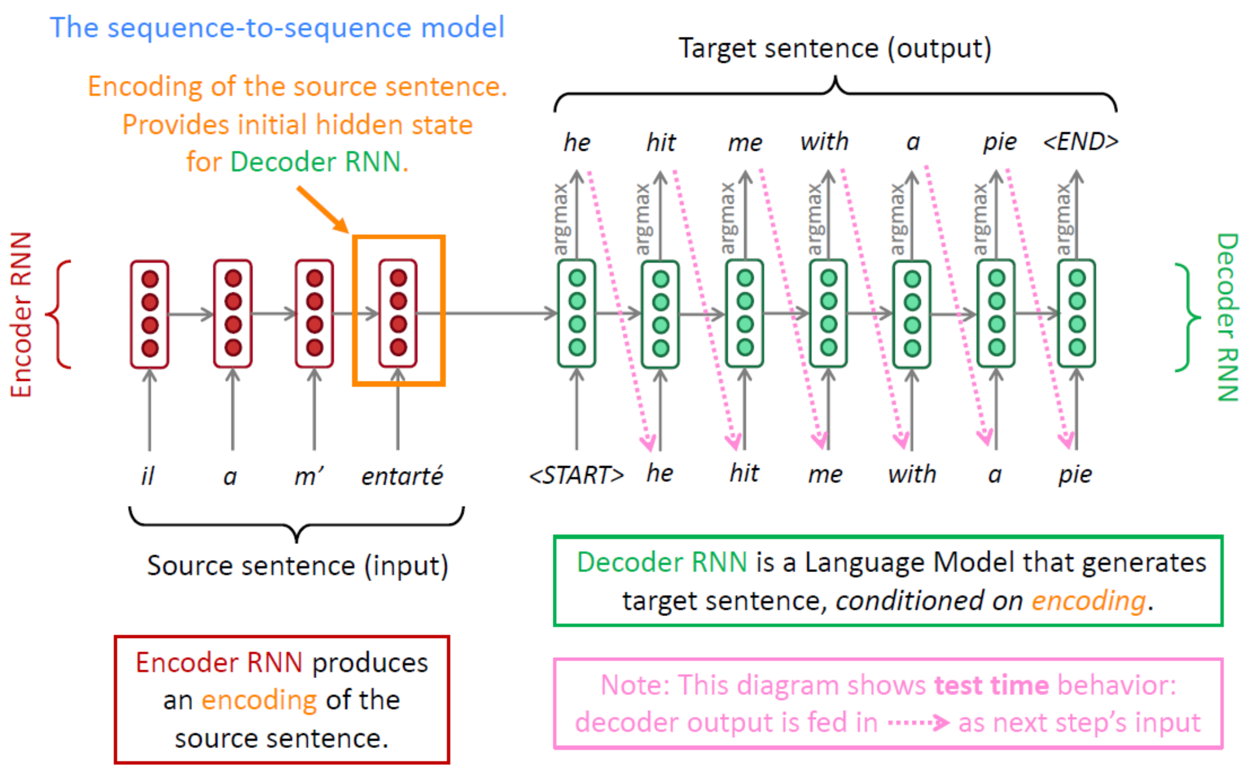

Figure 3.4: The test time inputs for the decoder. The output at the previous timestamp is used as the input of the current timestamp. The figure shows decoding using greedy decoding. Taken from Stanford's CS224N course.

The sequence to sequence system is optimized as a single system. Backpropagation operates end to end.

The problem with the Recurrent Neural Network shown above is the instability due to their recurrent nature. The gradient calculated using Backpropagation can either explode or vanish based on the magnitude of weights due to the repetitive multiplication of these weights across timestamps. For a complete proof sketch please look at [31] section 10.7. These problems were first mentioned in the paper "On the Difficulty of Training a Recurrent Neural Network" [42].

The Explosive gradient refers to the problem of having high magnitudes of gradients of loss with respect to the parameters of the model. These high gradients can make an update step too large and can cause our model to reach a bad configuration with a large loss.

A possible solution to the explosive gradient problem is the gradient clipping. Which scales down the gradients, if they become greater than some threshold before using them in the update step.

Algorithm 2: pseudo-code for norm clipping

$\hat{g} \leftarrow \frac{\partial \varepsilon}{\partial \theta}$
if $|| \hat{g} || \geq threshhold$ **then**
$\hat{g} \leftarrow \frac{threshhold}{||\hat{g}||} \hat{g}$
end if

The Vanishing Gradient problem refers to the problem of having a reduced value of the gradient with respect to the parameters of the model. The vanishing gradient is a problem because it leads to a very small update of parameters which hinders our learning process. In the case of RNNs, the small gradient obstructs the process of learning long term dependencies. This makes it too difficult for the RNNs to learn to preserve information over many timestamps.

Vanishing Gradient is a general problem in deep learning and is generally solved using skip connections across the layers of networks for example Resnet employs residual networks to solve this problem.

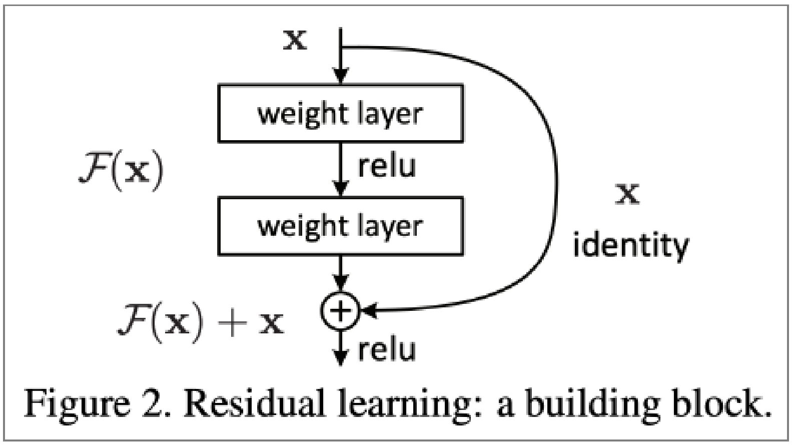

Figure 3.5: Taken from [5].

In a similar manner, RNNs also use something similar in the form of LSTMs and GRUs

Long Short Term Memory (LSTM)

Introduced in 1997 by [Hochreiter and Schmidhuber] [13] as a solution to the vanishing gradient problem.

The LSTM networks are the RNNs with explicit memory in the form of cell state alongside the hidden state as in the case of vanilla RNNs.

On each timestamp, we have a hidden state and a cell state, both of these are vectors of the same length. The cell stores long term information. The LSTM models are designed in a way they can erase, write, and read information from the cell. To facilitate this, the LSTM networks consist of three gates that are: Forget Gate, Input Gate, and the Output Gate. The gates are also the vector of length the same as the hidden state or the hidden state. The values of the gates can be between 0 and 1. The gates are dynamic, their value is computed as per the current input. For each timestamp, the values are calculated as per figure 3.6.

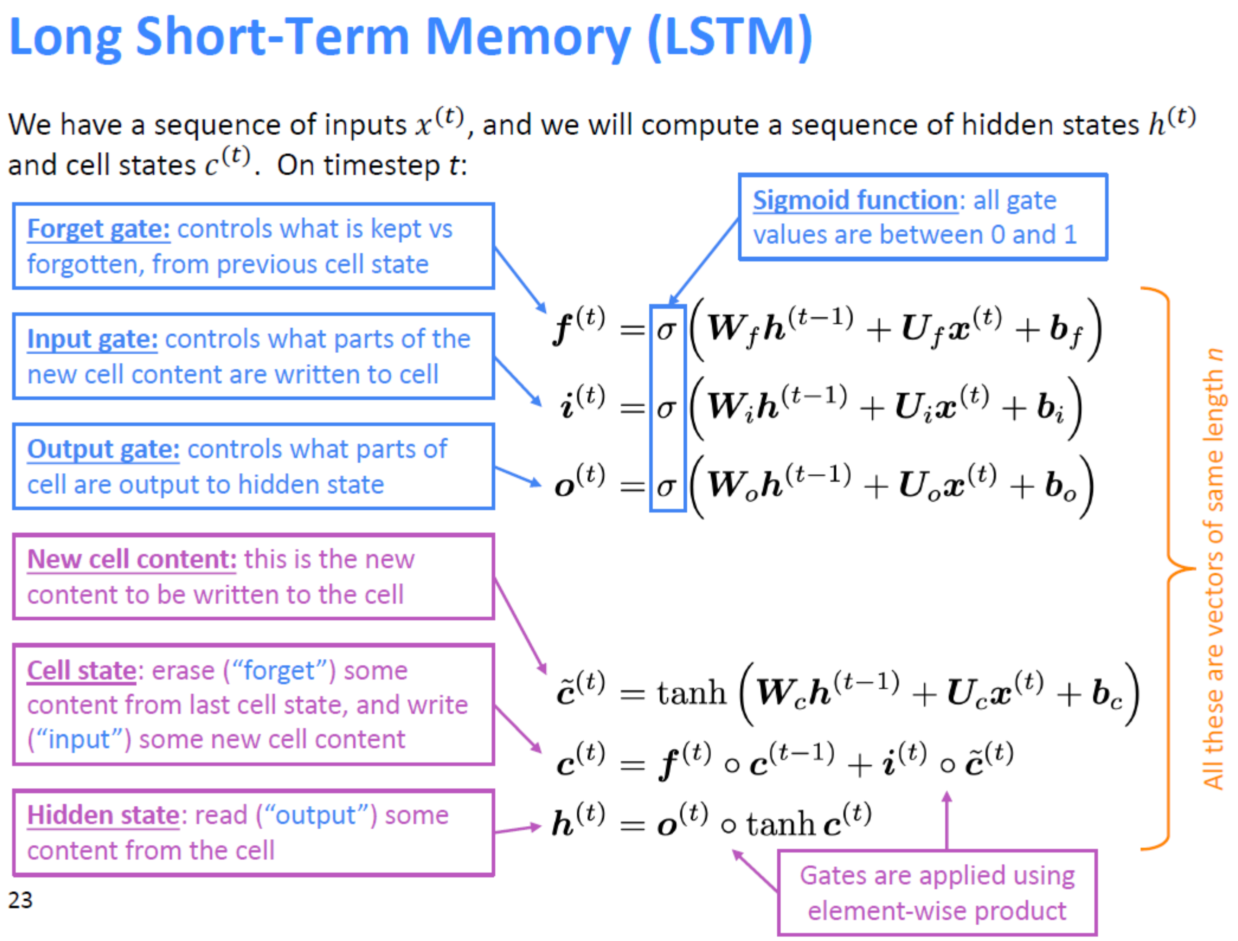

Figure 3.6: The order of computation of various gates, hidden state, cell state in the LSTM network. Taken from Stanford's CS224N.

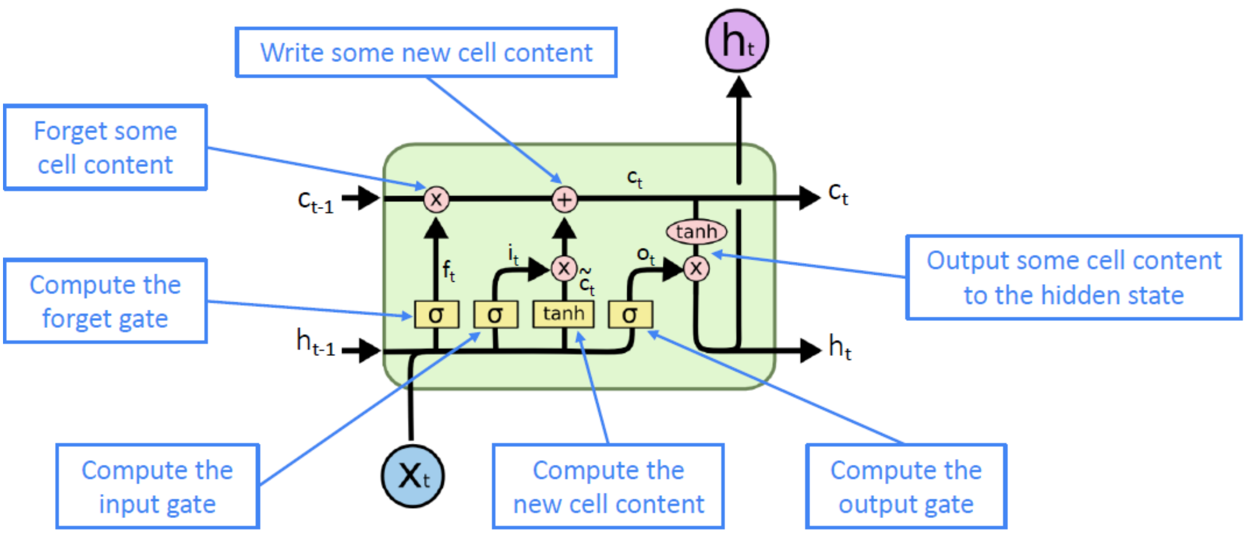

Figure 3.7: A look into the LSTM. Taken from colah's blog.

Gated Recurrent Unit(GRU)

A simpler alternative to the LSTM network proposed by [14]. In GRUs, we do not have an explicit memory cell state. We only have a hidden state and two gates Update gate and the Reset gate. Figure 3.8 shows the computation of various gates and hidden states.

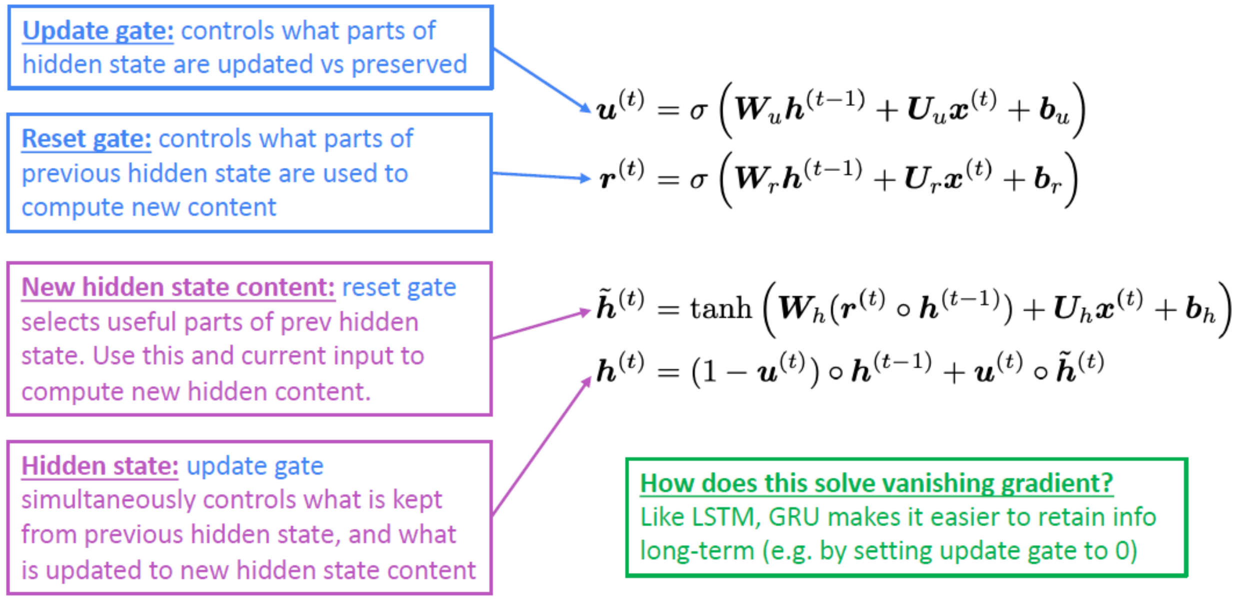

Figure 3.8: Computation of various gates and hidden states in Gated Recurrent Units. Taken from Stanford's CS224N course.

Recurrent Neural Networks can be bidirectional as well as have multiple layers stacked together.

Problem with the above architecture:

The last hidden state of the encoder needs to capture a lot of information about the input text and can lead to missing out on some important details from the input. This creates a bottleneck at the last hidden state as we have more information than space(Figure 3.9). One solution is to increase the size of the hidden state, but this solution does not scale up, as we want our RNN to process any length of the input, and deciding on the size is another problem.

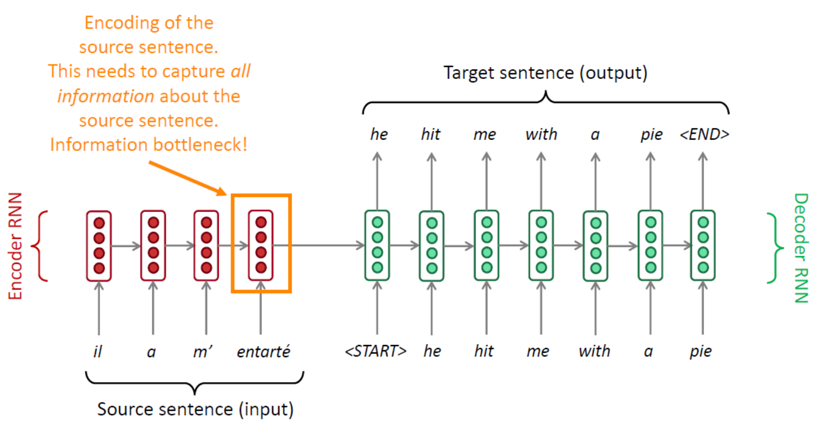

Figure 3.9: The bottleneck problem with sequence to sequence architecture. Taken from Stanford's CS224N course.

Solution: The Attention Mechanism

The Attention mechanism [2, 41] provides a solution to the bottleneck problem. The main idea is to use the decoder RNN at each step to directly interact with each of the encoder RNN steps so that while decoding each step can focus on any of the encoder's hidden states.

Mathematically,

Let hidden encoder states be $h_1, h_2, h_3, ..., h_N$ where $h_i \in R^n$

On kth timestamp, the decoder hidden state is $d(k) \in R^n$

$Attention\ scores = [f(h_1, d(k)), f(h_2, d(k)), f(h_3, d(k))...] \in R^N$

$Attention\ distribution(a) = softmax(Attention\ scores) \in R^N$

$Attention\ output = \sum_1^N (a_i) * (h_i) \in R^n$

The function f can be any function from below:

- Dot Product: $(h_1^T)d(k)$
- General : $(h_1^T)Wd(k)$
- Concat : $v^T(tanh(W[h1 : d(k)])$

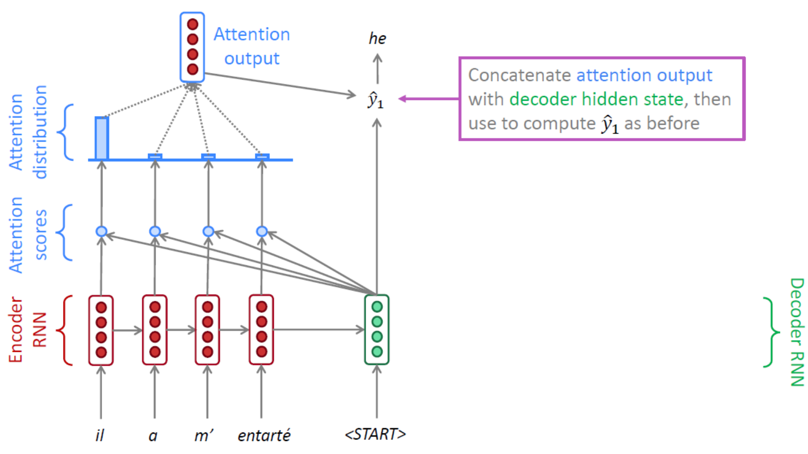

Figure 3.10: The decoder hidden state vector at each timestamp takes a dot product with all the encoder hidden states vector to generate Attention Scores which are all scalar. We then take softmax of these to get attention distribution. Encoder hidden states are scaled by the attention distribution scores and added to generate attention output which is concatenated by the decoder hidden state of that timestamp to generate output as done before. Taken from Stanford's CS224N course.

The attention mechanism also provides some relief from the vanishing gradients by providing bypass or skip connections. It also provides interpretability across the architecture as now we can infer to which part the decoder was focussing on each step of decoding.

The RNNs that we used for the above model needs to compute input sentences sequentially as the hidden state on timestamp t depends on previous hidden state at timestamp t-1, this makes their computation slow. RNNs prevent parallelization and lead to slower training and testing.

Transformer Model

The transformer model([3]) is also based on the sequence to sequence architecture but employs different modeling of encoder and decoder modules. The modeling allows parallelization of inputs in contrast with the sequence to sequence models based on the recurrent neural networks which hinder the parallelization of inputs.

The transformer model also contains two subparts namely encoder and decoder. The encoder and decoder can contain multiple stacked layers Figure 3.6.

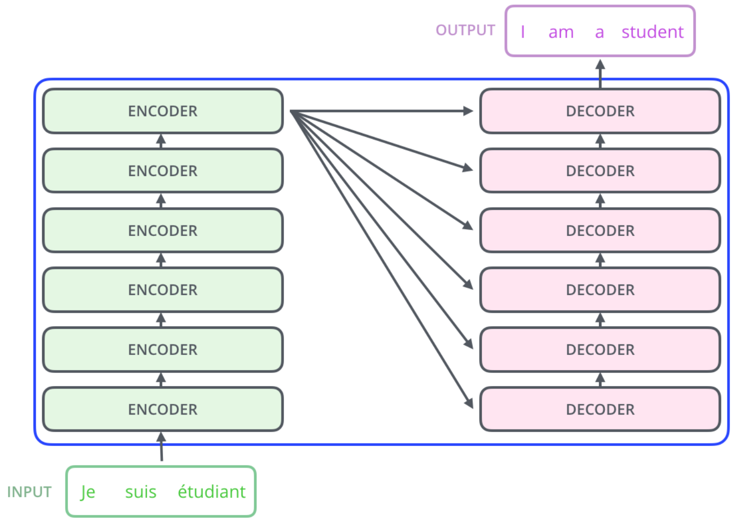

Figure 3.11: The Transformer model can contain multiple units of encoder and decoder modules. The original transformer proposed by [3] consists of six of each of the encoder and decoder. The number of encoders or decoders we use in our model is one of the hyperparameters that we set before training.Taken from Jalammar's blog.

The Transformer Encoder

The multiple encoders are all similar in architecture but do not share any weights. Each of the encoders consists of two sublayers 1. Self Attention Layer 2. Feed-Forward Neural Network.

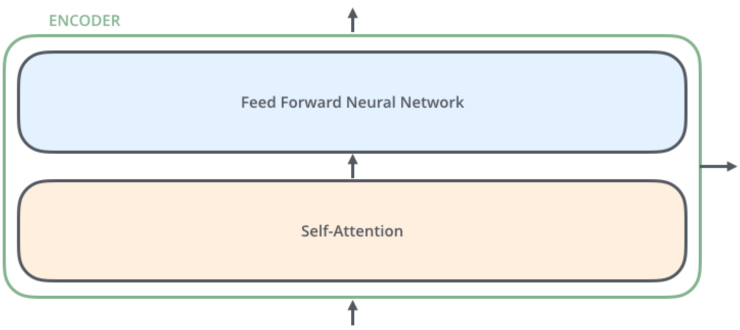

Figure 3.12: The Transformer Encoder consists of two sublayers.Taken from Jalammar's blog.

Self-Attention

The attention mechanism can be as mapping a query and a set of key-value pairs to an output. The output is calculated as a weighted sum of scores obtained from query-key pairs similar to the one we used in the previous model.

The self-attention in the encoder module consists of multi-head attention that consists of several attention layers running in parallel.

The input is passed as the query, key, and value. Each of the query, key, and value is passed through a linear layer of size (d_model,d_model) where d_model is the size of the embedding. Each of the query, key, and value after being passed through the linear layer is divided into multiple vectors, for example, a 512 query vector is divided into 8 such vectors of 64 dimensions each. These divided vectors are then used for multiple parallel scaled dot product attention(Figure 3.13). These divided vectors are then combined and passed through a linear layer.

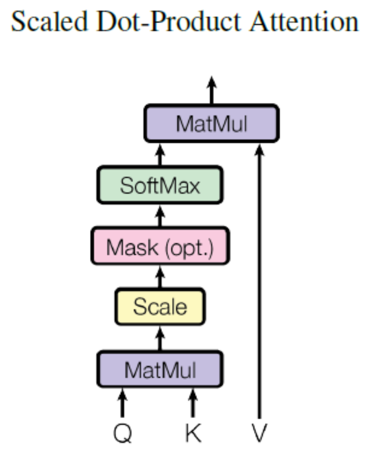

Figure 3.13: The Scaled Dot product in the Self Attention Layer of the Encoder. Taken from [3].

Feed-Forward Neural Network

This sublayer consists of a two-layer neural network with ReLu non-linearity in between.

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$$

The first layer is of dimension (d_model,d_ff) and the second layer is of (d_ff,d_model). In the original model proposed by [3], d_ff=2048 and d_model.

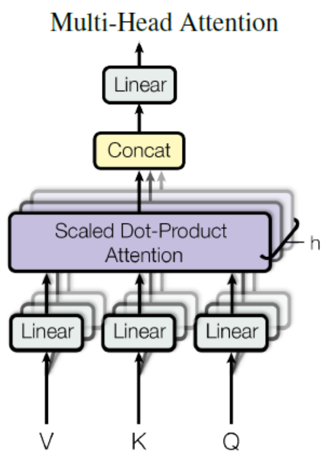

Figure 3.14: The Self Attention layer in Transformer Encoder Consist of Multi-head Attention Layer. Taken from [3].

The Transformer Decoder

The decoder consists of three sublayers:

1. A masked Multi-Head Attention Layer.
2. A Multi-Head Attention Layer
3. Feed-Forward Network

The masked Multi-Head Attention Layer takes the masked target during training and the decoder's prediction during testing similar to RNN based encoder-decoder.

Positional Encoding

The RNN based Sequence to Sequence based models can learn about the specific position of each word easily because of the serial computation of RNN's hidden state. However, because of the parallelization in the Transformer model, the sense of the position is somewhat not modeled explicitly. To model this, we add Positional Encoding to each embedding of the words. In the equation pos is the position of the word in the sentence and i is the dimension.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

We have described the architectures to very good lengths, however, we have left out some theoretical details about some of the things. For an in-depth review of these topics, please see the following 1. [31] for Recurrent Neural Networks 2. [11] for the Sequence to Sequence architecture 3. [2, 41] for attention mechanism applied to an encoder-decoder. 4.[3] for Transformer Model.

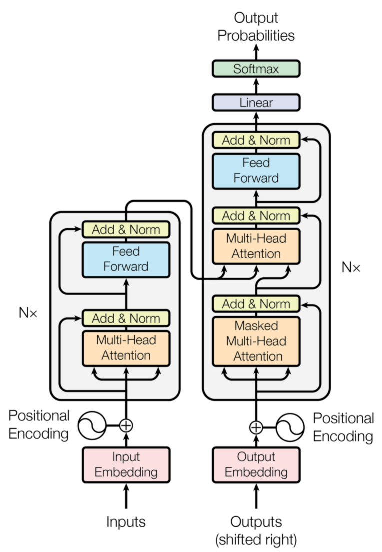

Figure 3.15: The Transformer Model. Taken from [3].

3.3 Decoding Algorithms

We have discussed above how the decoder works during training, its inputs are the target words tokens but are right-shifted. During testing, we don't have the target word tokens, so for the generation of the output text, we follow some decoding algorithms.

Greedy Decoding

For the first word token prediction, we take in a start token<START> as the input to the decoder. The decoder unit at this timestamp generates a probability distribution of words that might come next. In Greedy Decoding, we take the word with maximum probability to be the predicted word at this timestamp. We put the word token on to the next decoder unit as the input and the process goes on until we get <END> token or we have predicted some pre-defined number of tokens.

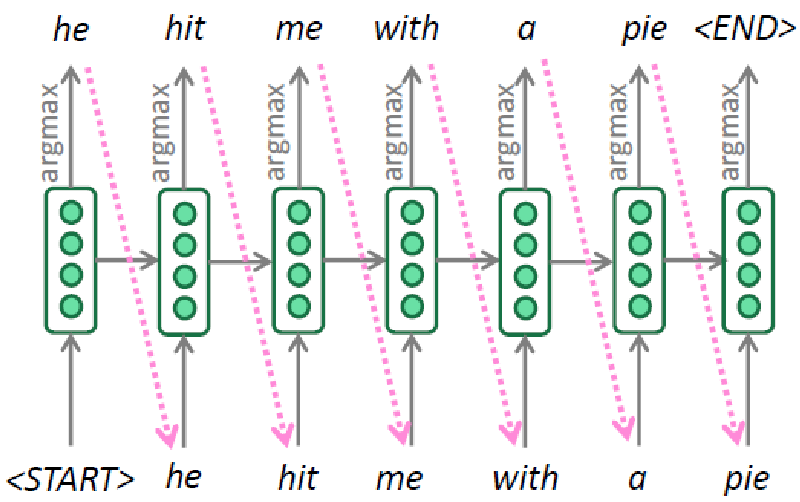

Figure 3.16: The greedy decoding simply takes the word token with the maximum probability at each timestamp. Taken from Stanford's CS224N.

The problem with the greedy decoding algorithm is that it cannot take back any of its predicted words, that is it cannot tract back to correct any mistake it did during previous decodings. The output can be nonsensical, unnatural, etc.

For example, it might generate something like he are an engineer instead of he is an engineer. One obvious solution to this is to analyze all the (V^K)[upto Kth timestamp, V is the size of the vocabulary] possible branches. The V^K is too large to be considered for analysis. This technique is referred to as the Exhaustive Search.

Beam Search Decoding

The idea of the beam search decoding is to reduce the hypothesis space of the exhaustive search by truncating most of the V^K branches.

On every timestamp of the decoder, we keep only the top t most probable partial predictions. Beam Search is much more efficient than the exhaustive search. The t is referred to as the beamwidth.

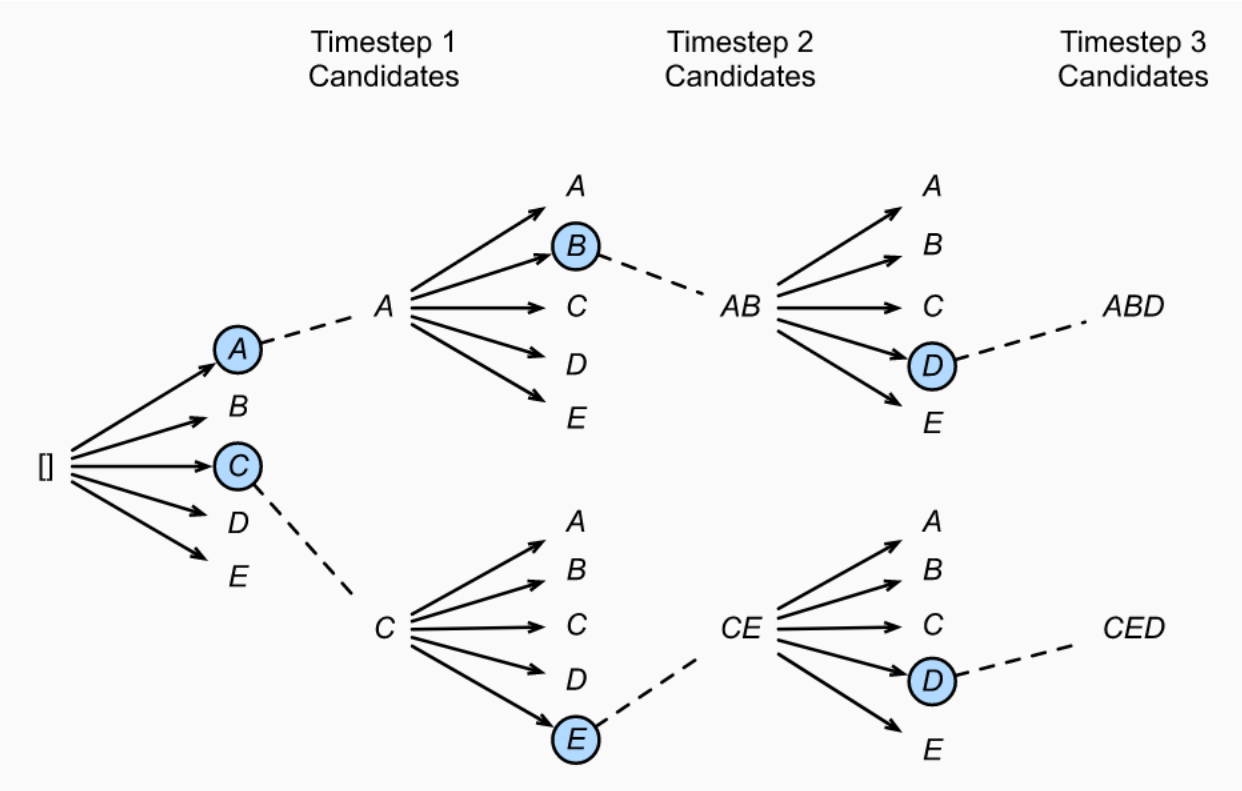

Figure 3.17: Beam Search. Taken from d2l.ai.

A smaller value of t generally produces sentences similar to ones produced by greedy decoding. A large value of t leads to generic responses and is generally less efficient. For t=1 it is greedy decoding for t=V(size of the vocabulary) its exhaustive search. Beam search has been a very popular algorithm used in many pieces of research regarding natural language generation Wiseman, Li. However, recent studies have shown that output produced by the beam search is more grammatical and low diversified leading to low-quality text generation [26, 43].

Top-k Sampling

The idea of this algorithm is to add a flavor of diversity/randomness in the output generated by a model.

On each of the timestamps of the decoder, randomly sample a token from the top k most probable tokens. If k=V(size of the vocabulary) it is known as Pure Sampling.

It has been shown that choosing an optimal k can be a difficult task, as sometimes the probability mass is evenly distributed and sometimes it is sparsely distributed[44].

To make the probability distribution smooth or jagged, a technique called Softmax Temperature is applied. The idea is to scale up or down all the probabilities. At a certain timestamp, we calculate the probability of a word by:

$$P_t(w) = \frac{exp(S_w)}{\Sigma_{w' \in V} exp(S_{w'})}$$

Applying Softmax Temperature to this makes it:

$$P_t(w) = \frac{exp(S_w/\tau)}{\Sigma_{w' \in V} exp(S_{w'}/\tau)}$$

Increasing temperature makes the output more diverse and probability distribution more smooth. Decreasing temperature makes the output less diverse and probability distribution jagged.

Setting the value of temperature between [0,1) with top-k sampling can greatly improve many of the shortcomings of the top-k sampling, as it shifts the majority mass of probability distribution in the top-k samples[26, 45]. However, as discussed it reduces diversity [46, 47].

Nucleus Sampling (Top-p Sampling) [44]

The idea is to truncate the probability distribution for the next word at an appropriate level. On each step of the decoder, we sample randomly from a set of words/tokens defined by The smallest set of words belonging to the vocabulary such that following holds:

$$\sum_{x \in V^{(p)}} P(x \mid x_{1:i-1}) \geq p.$$

We then scale the probability distribution as:

$$P'(x \mid x_{1:i-1}) = \begin{cases} P(x \mid x_{1:i-1})/p' & \text{if} x \in V^p \\ 0 & \text{otherwise} \end{cases}$$

where, $p' = \sum_{x \in V^p} P(x \mid x_{1:i-1})$

This method has so far been found the best among the ones described above[44]. We have experimented with all the decoding algorithms described here qualitatively.

3.4 Experiments specifications

We experimented mainly with two models 1. A sequence to sequence-based on Recurrent Neural Networks (LSTM and GRUs), 2. A sequence to sequence model on the Transformer model. We trained our model as the single-turn dialog system. Meaning, we do not take into account the previous utterances in order to predict the current utterance.

Dataset Used-Cornell Movie-Dialogs Corpus

In our experiments in this chapter, we used the popular Cornell movie Dialogs Corpus [48]. The corpus contains a large metadata-rich collection of fictional conversations collected from raw movie scripts. The dataset was collected from raw scripts and IMDB automatically and manually. Figure 3.1 shows some simple conversation snippets.

The corpus includes 220,579 conversational exchanges between 10,292 pairs of movie characters, involving 9,035 characters from 617 movies, a total of 304,713 utterances. Movie metadata that was included with the dataset included genres, release year, IMDB rating, number of IMDB votes, IMDB rating. Character metadata included gender and position on movie credit.

Before inputting the dataset for our model, we performed several preprocessing to suit our cause and limit. This preprocessing included extracting only the meaningful parts of the dataset, making a vocabulary out of the words used in the dataset, etc. For more details regarding the preprocessing please see appendix at the end of this chapter.

```
L1045 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ They do not!

L1044 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ They do to!

L985 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ I hope so.

L984 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ She okay?

L925 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ Let's go.

L924 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ Wow
```

Figure 3.18: Raw lines from movie_-lines.txt included in the dataset between two characters.

Training

Software

We used python 3.x to write all our code. Alongside python, we used the PyTorch framework for our models. Pytorch provides support to execute our models on GPUs [Graphical Processing Units] easily. We also have many other libraries for the computations involving preprocessing of data, reading text files, etc.

Hardware

We used freely available Google Collaboratory Notebooks for training our model. The Google Collaboratory Notebook offers random hardware for training the models. We used K80, Tesla T4, Tesla P4, Tesla P100 for training our models.

For more information regarding training please see the appendix section at the end.

Evaluation-Metric

Perplexity

A standard evaluation metric for Language Modelling task.

$$Perplexity = \prod_{t=1}^{T} \left(\frac{1}{P_{LM}(x^{t+1} \mid x^{(t)}, \ldots, x^{(1)})} \right)^{(1/T)}$$

This is equal to the exponential of cross-entropy loss. We require lower perplexity as it increases the probability of an actual sentence.

$$\prod_{t=1}^{T}\left(\frac{1}{\hat{y}^{(t)}_{x_{t+1}}}\right)^{1/T} = exp\left(\frac{1}{T}\sum_{t=1}^{T} -log\,\hat{y}^{(t)}_{x_{t+1}}\right) = exp(J(\theta))$$

Unigram F1

Metric used generally to evaluate the machine translation(MT) task. It is defined as the harmonic mean between unigram precision and unigram recall.

Unigram Precision(P): Fraction of words in the transformed sentence that are present in the actual sentence.

Unigram Recall(R): Fractions of words in the actual sentence are present in the transformed sentence.

P=m/T; P=m/A

F1=(2*P*R)/(P+R)

Loss Function Used

We used the standard cross-entropy loss function for both of our models.

$$H(p,q) = -\sum_{c=1}^{C} p(c)\,log\,q(c)$$

Where p is the correct probability distribution and q is the predicted probability distribution.

3.5 Results and Observations

We trained two models.

1. Sequence to Sequence based on RNN with Attention Layer
2. Sequence to Sequence based on Transformer model.

Sequence to Sequence based on RNN with Attention Layer

We trained sequence to sequence based on a two-layer encoder-decoder bidirectional Gated Recurrent Unit with attention mechanism for mapping dialogue responses. In total, the model had 14934316 trainable parameters. Hyperparameter settings can be found in table 3.1.

We trained our model for 200 epochs with checkpoint savings at every 10th epoch. Here, epoch refers to the model looking across all the training samples once. We used Adam optimizer [38] with default settings.

Hyperparameter	Setting
Attention Model	Dot Product
Number of Encoders	2
Number of Decoders	2
Embedding Size	500
Dropout Probability(Encoder)	0.1
Dropout Probability(Decoder)	0.1
Gradient Clip	50.0
Teacher Forcing	True (for training)
Learning Rate(Encoder)	0.0001
Learning Rate(Decoder)	0.0005
Batch Size	10
Number of Batches	4000

Table 3.1: Hyperparameter settings for our seq2seq with attention model.

Sequence to Sequence Based on Transformer Model

Hyperparameter	Setting
Number of encoder-decoder	2
Embedding Size	256
d_ff(Size of hidden layer in feedforward network)	512
Number of Multi-Head Attention heads	8
Dropout Probability	0.1
Batch Size	10
Number of Batches	4000
Learning Rate	0.00001
Betas in Optimiser	(0.9,0.98)
Eps in optimizer	10^{-9}

Table 3.2: Hyperparameter setting for the seq2seq based on the Transformer model.

We trained the model for 200 epochs with checkpoint saving at every 5 epochs. Here we used the Adam optimizer as above with the above setting.

The Transformer model outperformed the Recurrent Neural Network by a large margin as is evident from Table 3.3 and Table 3.4. The Recurrent Neural Network-based seq2seq model despite having 1.7x parameters as compared to the Transformer model cannot perform at par with the Transformer Model.

Model	Unigram F1	Perplexity
RNN based seq2seq	0.0244	2876959948.0777755
Transformer based seq2seq	0.5240	265667.2897

Table 3.3: Results for two of our models. These results are computed over the test set using greedy decoding.

We also observed the two models by manually chatting with them for a long time.

The sequence to sequence based on the Recurrent Neural Networks model produced way too nonsensical responses. Like when we inputted "What do you work ?" we expected something like "I am a programmer" but the model responded with "I ain't a dream".

Chatting with the Transformer model was up to the mark what we expect from a single turn conversational model. The responses almost every time do not feel out of the context. However, problems like repetition,

Model	Unigram F1	Perplexity
RNN based seq2seq	0.0887	38034340.2348
Transformer based seq2seq	0.5943	114691.363

Table 3.4: Results for two of our models. These results are obtained over all of the data including both training and test dataset.

genericness, response-relatedness remain. But we feel that for a single turn conversational model learning this much is very much appreciable. The responses were almost grammatical and have uniqueness in Figure 3.12.

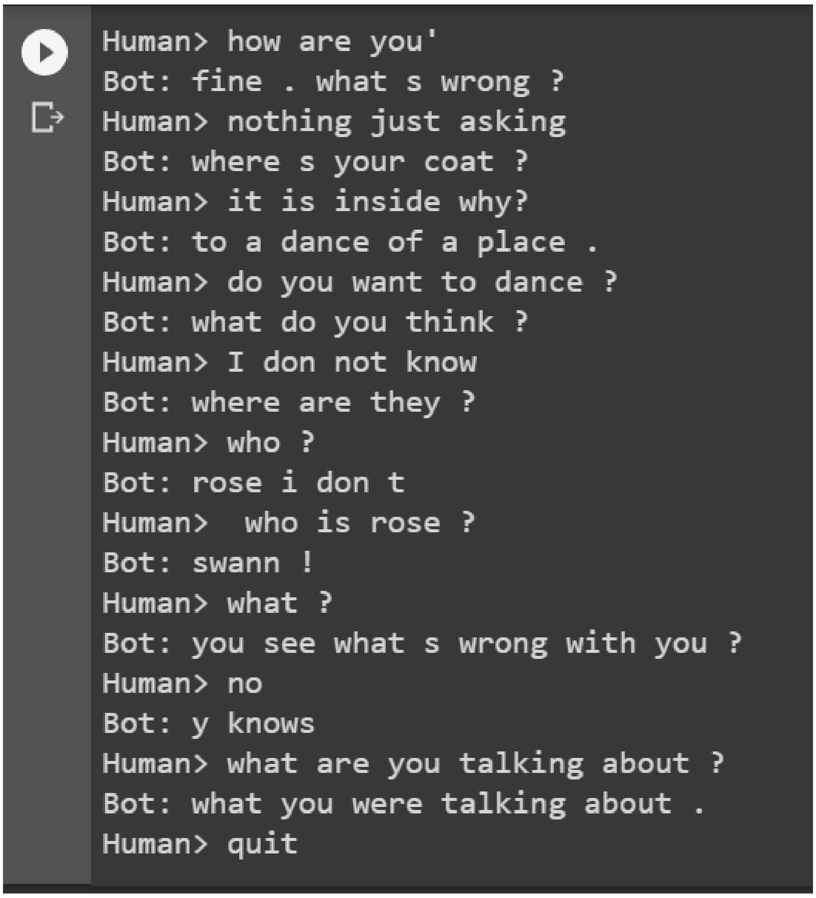

Figure 3.19: Conversation Snippet for the RNN based seq2seq model. Responses are decoded using the greedy decoding algorithm. The model mostly produced nonsensical outputs.

We also compared the efficiency of both the models by running an example of a batch size of 5 and the number of batches 4 (20 samples per epoch) and since the GPU assignment over the Google-Collaboratory is random we conducted this test locally on our own computer with GeForce 940MX.

Model	Parameters Count	Time Taken(seconds)
RNN based seq2seq	8903560	1.0515
Transformer based seq2seq	8647304	0.997

Table 3.5: Efficiency comparison between the run time of both the models.

We compared similar parametric sized models. The transformer seems to take less time but the parameter count for the RNN based is

```
Human: hello
Machine: hello .
Human: how do you do ?
Machine: do what ?
Human: how are you ?
Machine: fine .
Human: what do you do ?
Machine: i m going to kill him .
Human: do you have any friends ?
Machine: no .
Human: who do you work for ?
Machine: for a minute .
Human: I play guitar
Machine: that s right .
Human: yes
Machine: what did you say ?
Human: you are really a good person
Machine: yeah .
Human: where do you live
Machine: you re right .
Human: where do you live ?
Machine: in the morning .
Human: quit
```

Figure 3.20: Conversation snippet from seq2seq based on the Transformer model. The responses were somewhat better as compared to RNN based. We can also observe due to the open-ended nature of these systems they can lead to offensive responses. Also, we can observe adding punctuation marks changes the responses.

also somewhat greater than the Transformer's. The transformer model has the ability to take multiple inputs parallelly which greatly increases its efficiency. For a detailed experiment please look in the Appendix.

3.6 Conclusion and Future Work

In this chapter, we studied the two most popular Natural Language Generation architectures. We conclude that the model used for the NLG task can have a profound effect on text generation quality based on automatic evaluation and the manual evaluation by conversing with the model. The Transformer model not only provides efficiency but also leads to better results. The chapter also argues that having better models is not enough, we require better evaluation strategies[Decoding Algorithms] as well to generate engaging, interesting, natural text. The chapter ends with the conclusion that a lot of better algorithms are yet to be discovered.

The future work includes devising methods to include past dialogues as well in the context in order to generate the current response. We need to be able to include these multiple sentences efficiently. Also, we would like to have the conversational system to have some general knowledge, some persona so that it can have consistent responses. We also would like to have the system to have some other abilities like question asking, non-generic responses, etc. In all, we want to create a system that can converse very human-like. vv

3.7 Appendix

Preprocessing

The preprocessing steps involved in our implementation of conversational models are as follows:

1. Loading lines and conversations from the dataset.
2. Separating these lines and conversations into context and its corresponding response. We now have query-response pairs or context-response pairs. This step involved removing unnecessary information like movie id, character id, etc from the data that we loaded in the first step.
3. Normalizing each sentence of the query-response pairs or context-response pairs. Normalizing involves converting each character to lowercase characters, removing non-letter characters except for basic punctuations.
4. Filter pairs: Neglecting all those pairs of query responses that contain greater than 10 words in either context or response.
5. Prepare Vocabulary: It is the process to go through the entire corpus and look for each word and assign each unique word a token.
6. Words that come in our corpus less than or equal to three times are removed and hence the corresponding context-response pair which contains these words is also removed.

After all these steps of preprocessing on our corpus, we were left with 53113 query-response pairs and a vocabulary size of 7816 words. Out of the 53113 query-response pairs we used 40000 for query-response pairs for training our models and 13113 query responses were used for testing the models.

Training

For training models, we used the freely available Google-Collaboratory service. Google Collaboratory is a freely provided service by Google for the researchers to use the high-end hardware provided for computation of machine learning and deep learning models.

On connecting via Google-Collaboratory, we are connected to a virtual computer or server on Google's end. This virtual computer or server has more primary memory as well as it provides cutting edge GPU support. For comparison, our best laptop had an Nvidia Geforce 940MX which has 384 Cuda cores and the least graded GPU provided by the Google-Collaboratory service has over 2496 cuda cores. The cores can be thought of as how much parallel computation one can offer.

Since Google Collaboratory connects to a random server, the hardware configuration differs from time to time but on a whole, we got the following GPUs: Tesla K80, Tesla P4, Tesla T4, Tesla P100.

GPU	CUDA Cores	Memory
GeForce 940MX	384	4 GB
Tesla K80	2496	12 GB
Tesla P4	2560	8 GB
Tesla T4	2560	16 GB
Tesla P100	3584	16 GB

Table 3.6: shows a comparison between different Graphical Processing Units.

Further Experiments

Timing Comparisons [Efficiency Experiment]

We consider two parts:

1. Keeping Sequence Length Constant but increasing the number of parameters of the model.
2. Increasing Sequence Length which eventually increases the number of parameters.

Keeping Sequence Length Constant and increasing number of parameters of the model

For this experiment, we kept the number of encoders to be equal to the number of decoders for both the models.

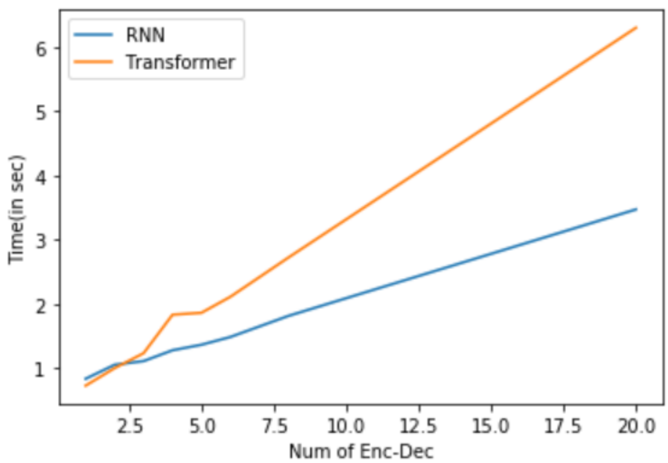

Figure 3.21: The graph shows how the time it takes to compute one epoch increases with parameters for both of the models.

As we can observe the time remains linear with the number of layers or number of parameters. We, however, observe that for the transformer model it increases more rapidly as compared to the RNN model.

Table 3.7: Number of parameters corresponding to the number of layers of encoder and decoder.

Number of Encoder, Decoders	Parameters(Transformer)	Parameters(RNN)
1	7329416	7326088
2	8647304	8903560
3	9965192	10481032
4	11283080	12058504
5	12600968	13635976
6	13918856	15213448
8	16554632	18368392
20	32369288	37298056

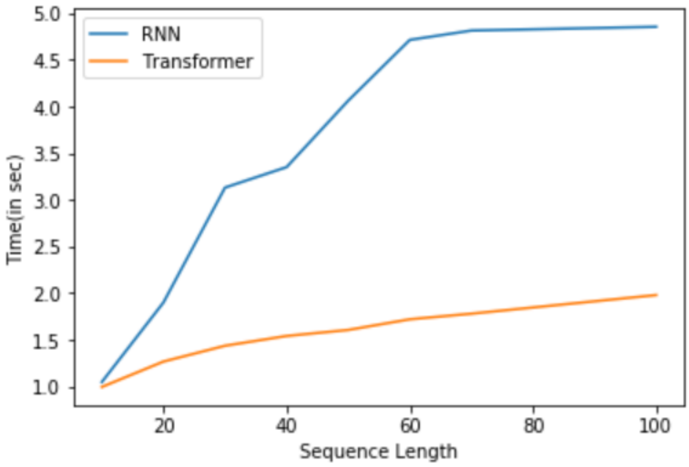

Figure 3.22: Time taken for the computation of one epoch as sequence length increases. Transformer based model had an almost flat variation with sequence length.

Increasing Sequence Length

In this experiment, we increase the length of sequences that are being given as the input to our models.

Since the transformer model is capable of processing multiple inputs at the same time, the increase in time for the increase in sequence length is way too small for it. This is one of the major advantages of the transformer model.

The sequential computation of inputs makes RNN based models slow which is evident from figure 3.14.

Experiments With Different Decoding Algorithms

We experimented with the following decoding algorithms:

1. Greedy Decoding
2. Beam Search
3. Top-k Sampling
4. Top-p Sampling or Nucleus Sampling

We used the sequence to sequence-based on Transformer models for these experiments

Greedy Decoding Due to the inability of the algorithm to go back to correct its mistake, the algorithm sometimes leads to nonsensical outputs.

```
Human: i study computer science
Machine: enough .
Human: why ?
Machine: because i m going to kill him .
Human: who are you going kill ?
Machine: i m going to get my name .
Human: okay
Machine: what s your name ?
Human: bianca
Machine: what s wrong ?
Human: nothing just talking to you
Machine: what s that ?
Human: nothing
Machine: what did you say ?
Human: i said nothing
Machine: you re a good man .
Human: yes i am
Machine: what are you doing ?
Human: making a report
Machine: we re going to be careful with him .
Human: with whom
Machine: it s a good idea .
Human: quit
```

Figure 3.23: Responses when the greedy-decoding algorithm is used. Sometimes the responses are ungrammatical, nonsensical.

Beam Search When beam search is used the output remains similar to the output given greedy-decoding algorithm for low values of beam width but as the beam width is increased the output starts becoming generic and hence generic(Figure 3.16,3.17,3.18) in most of the cases. Hence, the optimal value of the beam-width is very much necessary.

```
Human: hello
Machine: hello .
Human: how are you
Machine: okay .
Human: what are you doing ?
Machine: i m going to be a little bit .
Human: do you play any sport ?
Machine: no .
Human: i play football
Machine: and you have to do that ?
Human: no it is not that necessary
Machine: i know .
```

Figure 3.24: Decoded using beam search of beam width=5.

Top-k Sampling We talked at great lengths with the top-k sampled decoder, the reason was that it was diverse and do not repeat itself over the same input. The repetition avoidance made conversations very much

```
Human: i like working
Machine: what s your name ?
Human: jim
Machine: no it s okay .
Human: do you have any hobbies ?
Unkown Word..!!
Human: do you play any sport ?
Machine: no .
Human: okay'
Machine: what s your name ?
Human: jim
Machine: no it s okay .
Human: what is your name ?
Machine: i don t know .
Human: what do you do ?
Machine: i m going to kill him .
Human: how is the weather ?
Machine: good .
Human: do you like it ?
Machine: it s okay .
Human: q
```

Figure 3.25: Decoded using beam search of beam width=10.

```
Human: how do you do ?
Machine: do what ?
Human: how are you feeling ?
Machine: fine .
Human: I am feeling a little low
Machine: i don t want to talk about it .
Human: okay
Machine: what s your name ?
Human: jim
Machine: no it s okay .
Human: I love to dance
Machine: you don t want to .
Human: do you have any friends
Machine: i don t want to talk about this .
Human: what do you look like ?
Machine: like what ?
Human: when is this going to end ?
Machine: i don t know .
Human: i am stuck in my house
Machine: no it is
```

Figure 3.26: Decoded with beam search with beam width=20.

engaging and interesting. Below figures give some of the snippets.

```
Human: hi
Machine: hi .
Human: what are you doing
Machine: i m toward your stuff .
Human: okay
Machine: okay .
Human: okay
Machine: good night named grace .
Human: good night
Machine: good night
Human: how was your day
Machine: six months .
Human: I am very tired
Machine: are you ladies ?
Human: no
Machine: what is it ?
Human: nothing
Machine: seeing what ?
Human: quit
```

Figure 3.27: Output from pure sampling. The responses were very diverse and at times better than expected.

```
Human: hi
Machine: hi .
Human: how are you ?
Machine: good . i m okay .
Human: that's nice
Machine: yes .
Human: what are you doing
Machine: this is a long time .
Human: what are you doing ?
Machine: it s a living .
Human: do you play any sport ?
Machine: no .
Human: how was your day ?
Machine: fine .
Human: I am tired
Machine: are we going to be here ?
Human: quit
```

Figure 3.28: Decoded using top-k sampling with k=10.

Nucleus Sampling We find the output to be similar to the top-k sampling algorithm, but sometimes much diverse than the top-k sampling algorithm. Increasing k leads to better conversation.

```
Human: hi
Machine: hi .
Human: what do you do
Machine: i m going to get out .
Human: where do you live ?
Machine: i m in the room .
Human: okay
Machine: i ll take it .
Human: i am also in my room
Machine: is there a man ?
Human: yes why ?
Machine: i m in love with you .
Human: okay
Machine: you can t talk about it .
Human: sure
Machine: what s your name ?
Human: jim
Machine: yeah what s that ?
Human: quit
```

Figure 3.29: Decoded using top-k sampling with k=10 and temperatur=0.5.

```
Human: hello
Machine: hello .
Human: hi
Machine: hi .
Human: my mother is a doctor
Machine: thank you very much .
Human: what are you working on ?
Machine: nothing .
Human: how is the weather?
Machine: i m fine .
Human: when is this going to end ?
Machine: i don t know .
Human: okay
Machine: what do you mean ?
Human: nothing
Machine: what did you say ?
Human: quit
```

Figure 3.30: Decoded using top-k sampling methods using k=5 and temperature =0.5.

```
Human: hello
Machine: hi .
Human: what are you doing ?
Machine: i m sorry .
Human: which is you favourite song
Machine: it s why you been there .
Human: okay
Machine: you ll get it .
Human: whom do you live with ?
Machine: the store .
Human: where do you live ?
Machine: right here .
Human: who do you work for >
Machine: for seven will do you ?
Human: who do you work for ?
Machine: for a stupid friend .
Human: oh
Machine: what ?
Human: quit
```

Figure 3.31: Decoded using top-p method with p=0.5.

```
Human: hey
Machine: hey .
Human: do want to play guitar
Machine: right .
Human: which song do you like the most
Machine: you re till my heart .
Human: do you have any friends ?
Machine: no .
Human: I like cars very much
Machine: will you talk to me ?
Human: yes i will talk to you ?
Machine: for what ?
Human: anything
Machine: good .
Human: quit
```

Figure 3.32: Decoded using nucleus sampling with p=0.9.

The Context 4

4.1 Overview

Context forms one of the most integral parts of any language processing task whether it is representing words in the computer or predicting the next word that may come up in a piece of text. The context is important due to the fact that the new prediction is very much dependent on it. In the context of dialogue systems, the history of dialogues that happened to a point can be considered as the context for the next response by the system. The context sometimes can also be combined with some other sentences to aid with generating specific dialogues that contain specific traits that we desire. These traits can be a particular persona, certain knowledge sentences, etc.

In this chapter, we first discuss models that are designed specifically to include multiple sentences namely Hierarchical Recurrent Encoder-Decoder [21], we then specify a general method used generally in order to train such systems. We then discuss a case of how we can include knowledge explicitly in the case of the Neural Conversational Models.

4.2 Including Context

Hierarchical Recurrent Encoder-Decoder

The HRED for dialogue systems proposed here is based on one proposed in [49] which is further an extension of [21].

The problem is to model the history of dialogues as the hierarchical context in a single vector and generate a possible relevant response to it. The hierarchy can be referred to as the priority weights associated with each of the history dialogue that occurred to a particular moment. The semantics of recent dialogue should be more important than the one at the start.

The history of dialogues is considered at two levels - A sequence of utterances and a sequence of tokens comprising these utterances.

Consider having k utterances representing the history of dialogues. Each of k utterances generates an utterance-vector when mapped through an Encoder RNN. The utterance vector is the hidden state obtained after the processing of the last token of the sequence. Another encoder RNN process each of these utterance vectors iteratively. After processing the kth utterance vector, the hidden state of this encoder RNN represents a summary of all the k utterances. This hidden state is then used to decode the response for the system.

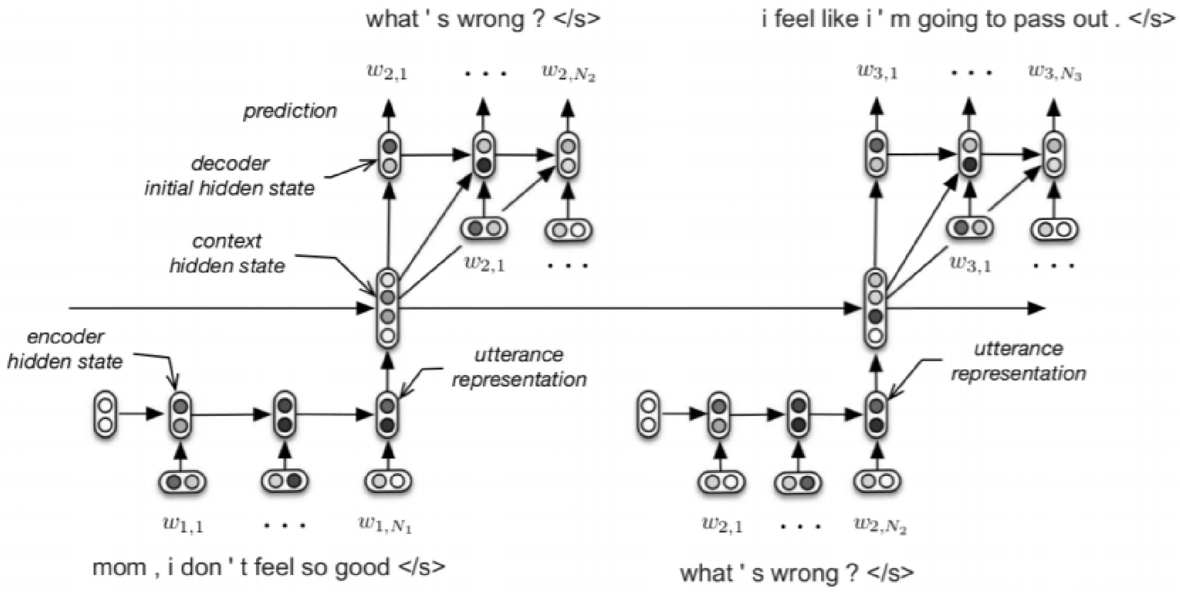

Figure 4.1: HRED architecture for a dialogue composed of three turns. Taken from [49].

General Method

A more general method that can apply to both Transformer based and RNN based models is to concatenate all the sentences of context together into a single sentence and then encode it using RNN or Transformer. Since RNN and Transformer are not dependent on the sequence length, the method is quite convenient. The method is also similar to the way language models are trained and therefore large enables us to use language models pre-trained on large corpora to be finetuned on the dialogue task.

4.3 Including Knowledge: Knowledge-Based Conversational Agents

In our previous chapter, we do not include any previous history of dialogues alongside our query, also we do not include any explicit knowledge in order to help our conversational system. We just hoped that our model would somehow learn the knowledge required by the user query through our dataset in order to answer it. Knowledge forms one of the most important parts of our conversation with automated answering, hence it is of utmost importance that we include knowledge sentences relevant to our context alongside our dialogue history.

The End2End model based on the Transformer model consists of one shared Transformer Encoder to encode context sentences and knowledge sentences. The encoded context sentence acts as the query and the encoded knowledge sentences act as the values for the attention mechanism. The resultant is concatenated with encoded context. This context concatenated tensor is passed through the decoder to generate the model's response(Figure 4.2).

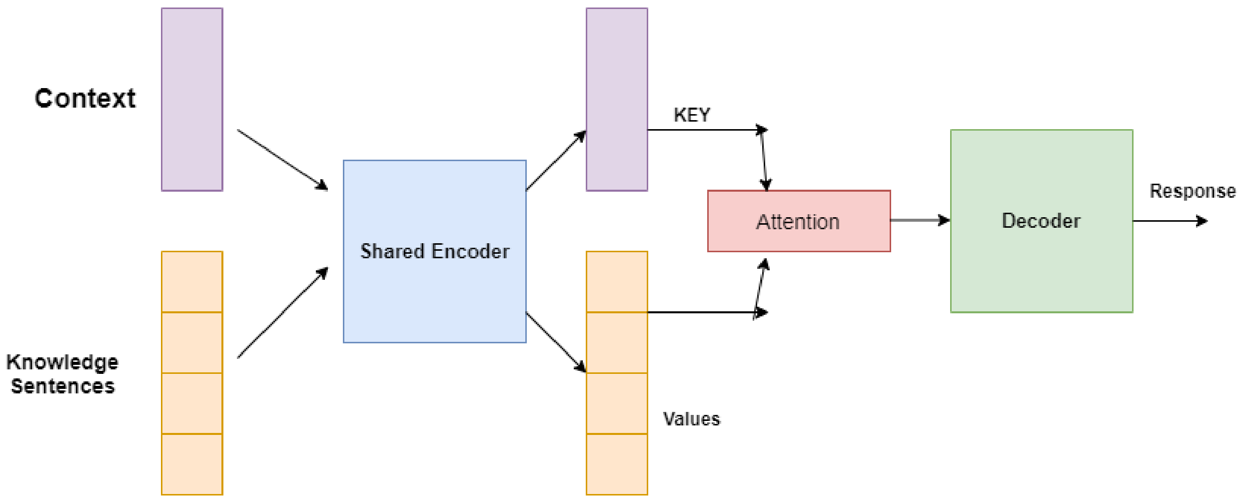

Figure 4.2: The End2End Generative model for including knowledge explicitly. Based on [23].

4.4 Dataset Used

We used the recently released Wizard of Wikipedia dataset for our work. The setting for the conversation involved two agents that involve themselves in the conversation with one of them selecting the starting topic of the conversation. The topic of the conversation is allowed to change naturally during the course of the conversation. The two agents are not symmetrical, one of them is a knowledgeable teacher who happens to have a large base of knowledge based on the context of the conversation beforehand and the other is an inquisitive learner. The knowledgeable agent is the machine and the inquisitive learner is the human. The flow of the conversation as described in the original paper [23] is as follows:

1. Either the wizard or the apprentice chooses the opening topic and speaks first. The conversation begins.
2. The wizard is given knowledge sentences based on the context of each turn, the wizard generates the utterance based on these sentences.
3. The conversation continues until one of them is decided to end.

The topics included such as podcasts, music festivals, etc. In total there were 1365 open domain crowdsourced dialogues. The final dataset contains 22311 dialogues with 201999 turns.

4.5 Results and Observations

We finetuned a pre-trained model based on Transformer architecture given in [23]. We evaluated the model on random split data of topics that contain both seen and unseen topics as well as on data containing unseen topics. The results of which with respect to automatic metric is given in Table 4.1 and Table 4.2.

Model	Perplexity	Time Taken(seconds)
E2E Generator Model (Dinan et al 2019 [23])	60.06	0.1734

Table 4.1: End to End Transformer based model's evaluation metric on random-split topics. Trained using the ParlAI framework.

Model	Perplexity	Time Taken(seconds)
E2E Generator Model (Dinan et al 2019[23])	103.1	0.1525

Table 4.2: End to End Transformer based model's evaluation metric on unseen topics. Trained using the ParlAI framework.

Due to the hardware limitations, we were unable to interact with the model itself. However, from the original paper, we looked at various conversations, some of them are given in the figures below:

Topic: Shower (*unseen*)
Human: So I've been looking for a better showerhead with more water pressure for my shower.
Model: I love pressure-versed water at nearly constant temperature to a shower
Human: That sounds pretty great! Love those rainshower showers!
Model: the ancient greeks were the first people to have showers.
Human: Well I knew they were good for something.
Model: the original showers were neither indoor structures nor man made but were common.
Human: Ah that sounds intimidating!
Model: it is required manual transport of both fresh and waste water.

Figure 4.3: Conversation Snippet from [23]. The model used for outputting the responses was the Two-Stage Transformer Model. As can be seen from the snippet including context (the history of dialogues) makes for a pleasant experience for the conversation partner.

Topic: Circus (*seen*)
Model: have you ever been to a circus? it is a company of performers who put on diverse entertainment shows.
Human: I went to the circus once when I was a child actually.
Model: that's great! entertainment is a form of activity that holds the attention and interest of an audience.
Human: That's true. Have you ever been to a circus?
Model: I have not, but I know they were built with various types of seating ring, and sometimes a stage.
Human: I went to one held in a stadium, so we had box seats.
Model: that's awesome! i'm sure you'll have to go to the circus, with appropriate music.
Human: Yes, circuses usually have their own kind of music.
Model: circuses can be done by circuses.

Figure 4.4: Conversation snippet from [23].

4.6 Conclusion and Future Works

Embedding context in our dialog systems greatly improves the system's dialogue responses, the responses are much more related to the current theme of the conversation and thus lead to deep conversations with the conversational partner. As the input has more entropy now (due to the inclusion of increasing context at every turn of the dialog), we need to design more sophisticated and efficient models. Including explicit knowledge at every turn of the dialog surely makes the model more versatile in communicating with humans but this explicit modeling of knowledge has its own cost-The Memory. We require memory to store these sentences and often very large memory is required to work upon. Including these into our system also increases the computation time of our model and we have to use faster machines in order to train and evaluate these systems.

Future works can include but not limited to 1. Factual Checking in the models using knowledge graphs or using some other mechanism

2. Embedding knowledge without actually having to have a separate knowledge base 3. To have these systems of sizes similar to their training data set, most of these models have parameters in the range of millions, while our training data is in thousands, this eventually will make more efficient and better models. The aim is to bring these thoughts together in order to make an efficient, knowledgeable, natural dialog system.

Adding Control | 5

5.1 Overview

Natural Language Generation(NLG) tasks like open-ended conversational agents, Abstractive Summarization, etc are more open-ended as these tasks can have a large body of correct outputs corresponding to a single input. This makes us focus our attention on attributes that really make some outputs better than others from the correct set of outputs. From our previous discussions, we can infer some of these attributes such as the ability of dialogue systems to remain stick with the current topic of conversation, ability to output different outputs when the same input is given multiple times. Other attributes can be the consistent nature of the system, question asking nature of the model, etc.

In this chapter, we discuss two main methodologies to control these attributes. We discuss how we can control four main major attributes Response-Relatedness, Question-Asking, Genericness, Repetition. We also discuss the framework for modeling these attributes, so that we can extend and include other attributes as well. We discuss how modeling these attributes affect our dialogue system with respect to automatic metric namely Perplexity and Unigram F1 score. We then finally conclude our discussion by discussing some future work that can be done to improve upon these.

5.2 Control Methods

Ability to make the output of our dialogue system to have certain attributes refers to controlling the output. As we can infer from our previous discussions, decoding algorithms can control the output of a neural conversational model. Decoding algorithms are used at the test time. So, one method can be is to change the way how we decode at the test time. Similarly, there exist methods that can be employed during training which can lead to an output having attributes which we desire.

Two methods that can be used for the addition of control are:

1. Conditional Training(CT)
2. Weighted Decoding(WD)

Conditional Training (CT)

Standard sequence to sequence models learns the probability distribution of output y given x i.e $P(y \mid x)$. In our case, y is the next utterance, and x can be the history of dialogue(s) or context. The conditional training model learns the probability distribution given x(context) and z, a discrete variable that signifies the desired output attribute. We can have multiple discrete variables to control various attributes that we wish to control. In that case, our probability becomes P(y | x,z1,z2,..).

Several researchers [24–26, 29] have different versions of z to include for various attributes.

Weighted Decoding(WD)

Weighted decoding is a decoding algorithm that increases or decreases the probability of words with certain features. Since the method is used to decode, it is applied only at the test time, requiring no change to the training methodology. This method can be applied in the presence of any of the decoding algorithms(Beam Search, Greedy Decoding, K-Top Sampling, Nucleus Sampling). Weighted Decoding can be used to control multiple attributes simultaneously and can be used alongside conditional training.

In weighted decoding, on the kth step of decoding, a score for each of the words in the vocabulary is calculated based on the partial hypothesis or the group of words that were predicted up to the (k-1)th step.

$$\text{score}(w, y_{\leq t}; x) = \text{score}(y_{\leq t}; x) + log\ P_{RNN}(w \mid y_{\leq t}, x) + \sum_i w_i * f_i(w; y_{\leq t}, x).$$

- $score(y_{\leq t}; x)$ Accumulated score of the partial hypothesis
- $logP(w \mid y_{\leq t}, x)$: The log of decoder probability of the word
- f_i: The feature of attribute assigns value to w based on context and partial hypothesis.
- W_i: The corresponding weight associated with the features. A positive weight increases the probability of word w that scores highly with fi. A negative weight decreases the probability of words that score highly with feature fi.

5.3 Repetition

From our previous experience, we can define the repetition as follows:

1. Repetition of utterances across inputs by the user.
2. Repetition across utterances.

We can design features to capture these repetitions and assign them with negative weights in the weighting decoding so as to reduce the repetition.

5.4 Specificity

Specificity refers to the ability of the model to output more specific responses and remain away from generic responses. For example in the 3rd chapter, we looked at how the model's response becomes more generic and boring on increasing beamwidth in beam search decoding.

We can associate scores for each word based on its occurrence so far in the responses. As per [29, 50], we can use Normalised Inverse Document Frequency[NIDF]. Which uses Inverse Document Frequency score for its computation. This is used along with the weighted decoding[WD].

Inverse Document Frequency=$log(R/C_{(w)})$

R: Number of responses so far. $C_{(w)}$: Number of responses containing word w.

Normalized Inverse Document Frequency is calculated as:

$NIDF_{(w)} = IDF_{(w)} - min\_idf/(max\_idf - min\_idf)$

5.5 Response Relatedness

In conversations, it is very much desirable to have utterance responses to be very similar to the partner's last utterance. From our previous experience when we input "How are you ?" the output results in "Have you had any before ?" which is very much not related to our response. For weighted decoding, we can use a similarity score such as given in [28]:

score = $cosine\_similarity(emb(w), sen\_emb(l))$

$sen\_emb(l)$: Average of glove embedding for every in the sentence.

5.6 Question-Asking

We want our model to ask questions as it provokes one to be engaged in the conversation. For weighting decoding, we can associate each word with a positive score if that word is among the [how, what, when, where, who, whom, etc].

The above attributes can also be modeled in a similar manner with conditional training(CT).

5.7 Results and Observations

We used ParlAI [51] a software platform for dialog research for observing the outputs obtained without control and with control added.

The baseline model is similar to the one used in [29], an LSTM bidirectional model pre-trained on twitter dataset, and then fine-tuned on the persona chat [22] dataset.

Baseline Model without any control

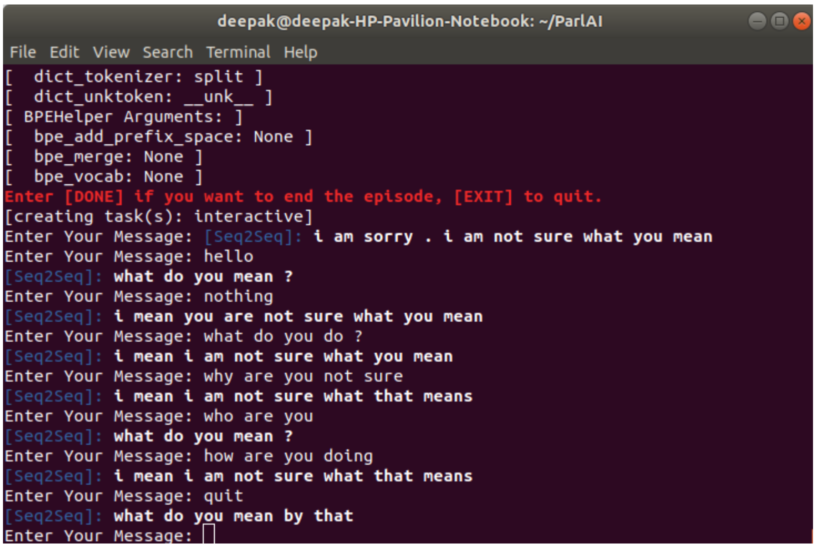

Figure 5.1: Decoded using the greedy decoding algorithm(or beam decoding with beam size=1)

We can observe due to the use of context as the history of all dialogue, the machine is able to somewhat remember the context. It stays on a single topic. We can also observe that the system repeats too much and this makes the model quite uninteresting. The model also asks a lot of questions and does not provide any prompt answers

Baseline Model without any control

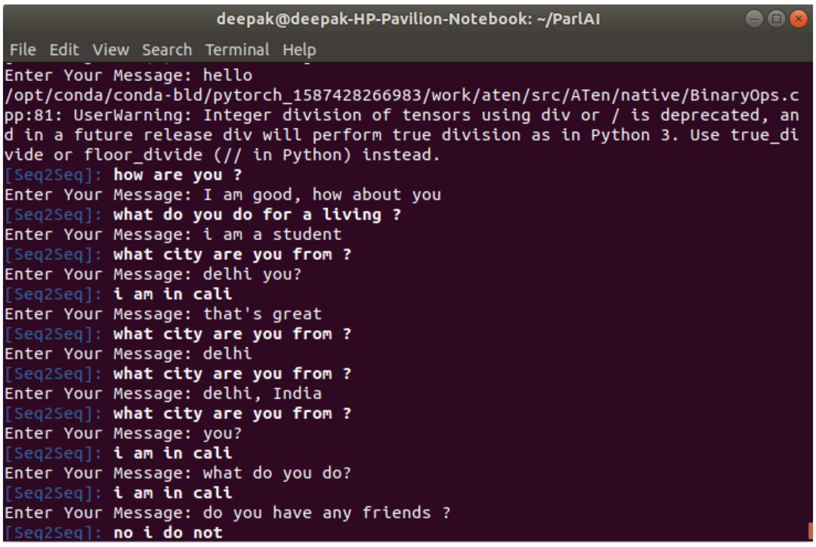

Figure 5.2: Decoded using beam search with a beamwidth of 20

The response is mostly similar to the previous setting. However, due to the higher beam size, results in genericness which leads to the same response to many of the inputs. There are also lots of unnecessary questions.

Repetition Controlled using Weighted Decoding

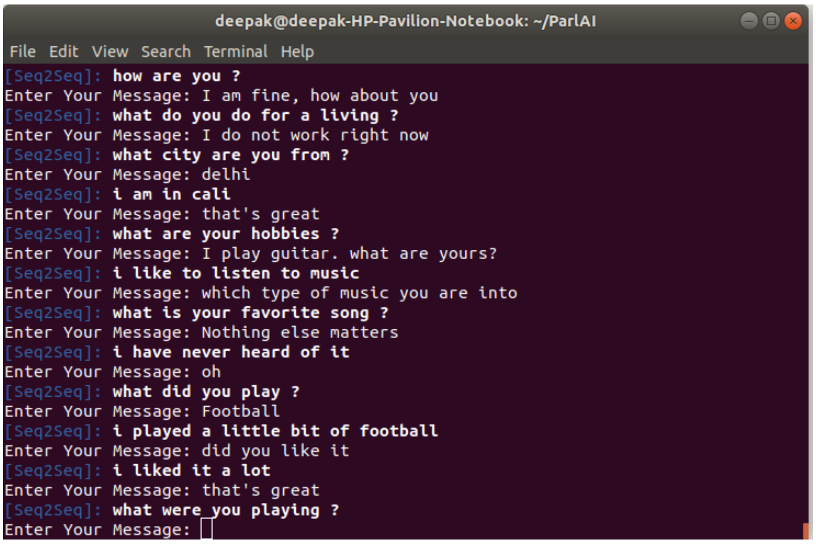

Figure 5.3: Decoded using beam search of beam size 20 alongside weighted decoding.

This one makes for a very interesting conversationalist at least in comparison to the above two. Since the output is decoded using weighted decoding to explicitly have less repetition, the output has less repetition which really improves the quality of conversation.

The repetition removal greatly improved the experience for us, so we will have the repetition removal attribute in each of the following cases.

Question Controlled through weighted decoding and repetition control through weighted decoding

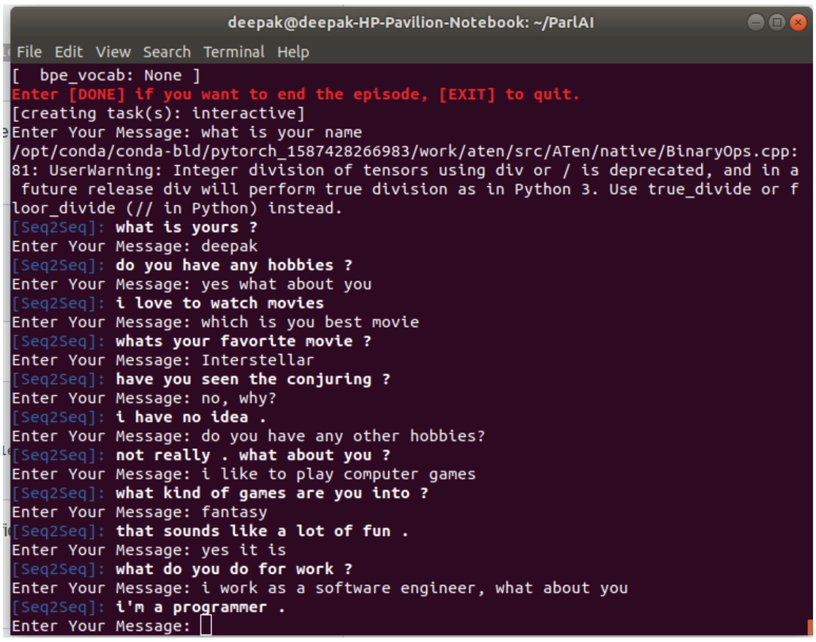

Figure 5.4: Conversation snippet for the model with the question asking control and repetition removal control.

The response for these models also seemed to be similar to the previous one. Change is not very subtle. We can also observe sometimes the response of the model is well inside the context, but it is not that too deep to make a human-like conversationalist.

Specificity Control Conditional Training(CT) and Repetition Control through Weighted Decoding(WD)

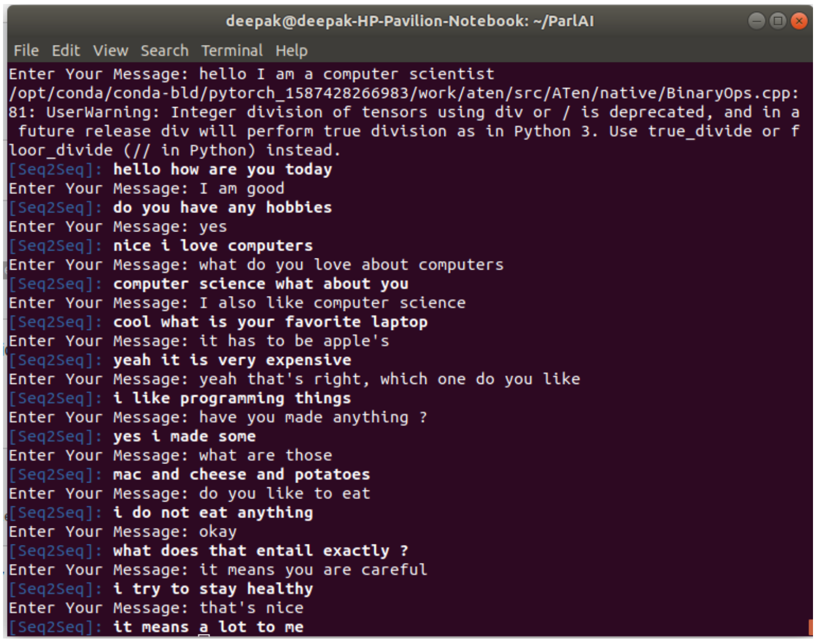

Figure 5.5: Conversation Snippet for the model with Specificity control using CT and repetition control using WD.

The responses are good but in case of some very deep context, the responses just fail. This indicates either the model or training data just fails to capture these scenarios. Also, while the bot remains attached to the context of the conversation, sometimes due to the open-ended nature of dialogue agents the answer or response might be correct for the input but we as a conversational partner do not like it.

Specificity Control using Weighted Decoding and Repetition Control using Weighted Decoding

We can observe some factual wrong information here as Fast and Furious movies are not super scary. When we train the open-ended model we do not generally care factual information to be output always correctly by the system, but we surely desire such a system which can answer factual right answers without having to explicitly include in them another for factual information only like Google Assistant or Amazon's Alexa uses. Work in this direction is one of the research topics.

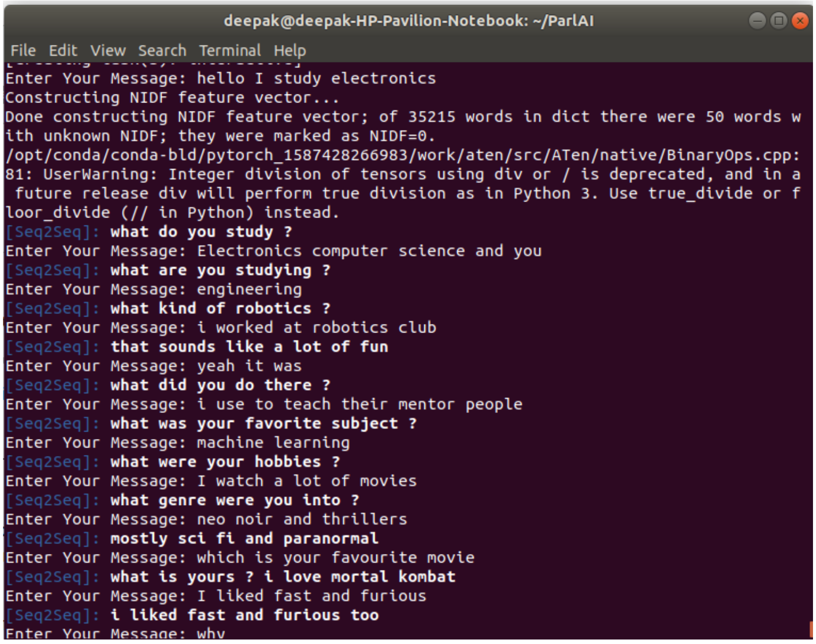
```
deepak@deepak-HP-Pavilion-Notebook: ~/ParlAI
File Edit View Search Terminal Help
Enter Your Message: hello I study electronics
Constructing NIDF feature vector...
Done constructing NIDF feature vector; of 35215 words in dict there were 50 words w
ith unknown NIDF; they were marked as NIDF=0.
/opt/conda/conda-bld/pytorch_1587428266983/work/aten/src/ATen/native/BinaryOps.cpp:
81: UserWarning: Integer division of tensors using div or / is deprecated, and in a
 future release div will perform true division as in Python 3. Use true_divide or f
loor_divide (// in Python) instead.
[Seq2Seq]: what do you study ?
Enter Your Message: Electronics computer science and you
[Seq2Seq]: what are you studying ?
Enter Your Message: engineering
[Seq2Seq]: what kind of robotics ?
Enter Your Message: i worked at robotics club
[Seq2Seq]: that sounds like a lot of fun
Enter Your Message: yeah it was
[Seq2Seq]: what did you do there ?
Enter Your Message: i use to teach their mentor people
[Seq2Seq]: what was your favorite subject ?
Enter Your Message: machine learning
[Seq2Seq]: what were your hobbies ?
Enter Your Message: I watch a lot of movies
[Seq2Seq]: what genre were you into ?
Enter Your Message: neo noir and thrillers
[Seq2Seq]: mostly sci fi and paranormal
Enter Your Message: which is your favourite movie
[Seq2Seq]: what is yours ? i love mortal kombat
Enter Your Message: I liked fast and furious
[Seq2Seq]: i liked fast and furious too
Enter Your Message: why
```

Figure 5.6: Conversational Snippet for the model with specificity controlled using WD and repetition control using controlled using WD.

Response Relatedness Control using Weighted Decoding and Repetition Control using Weighted Decoding

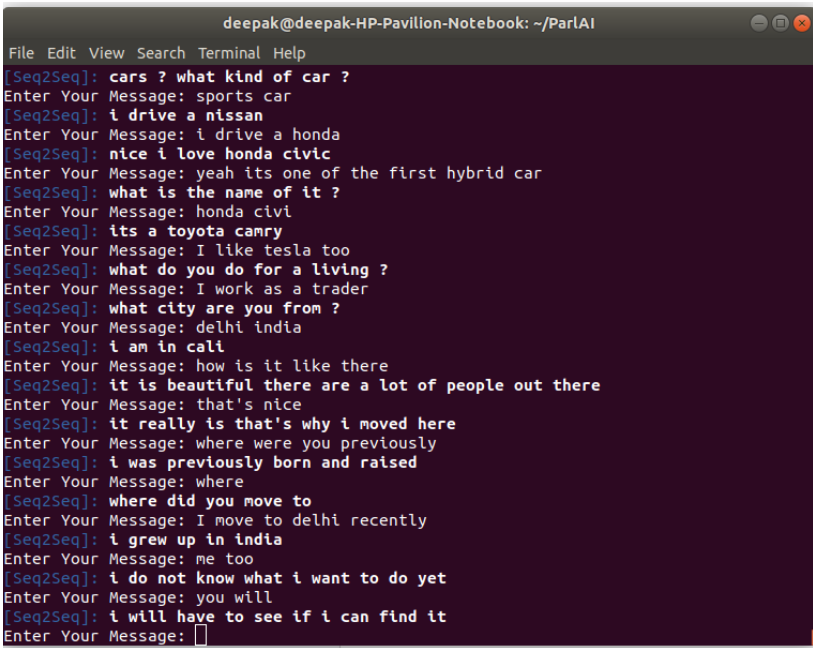
```
deepak@deepak-HP-Pavilion-Notebook: ~/ParlAI
File Edit View Search Terminal Help
[Seq2Seq]: cars ? what kind of car ?
Enter Your Message: sports car
[Seq2Seq]: i drive a nissan
Enter Your Message: i drive a honda
[Seq2Seq]: nice i love honda civic
Enter Your Message: yeah its one of the first hybrid car
[Seq2Seq]: what is the name of it ?
Enter Your Message: honda civi
[Seq2Seq]: its a toyota camry
Enter Your Message: I like tesla too
[Seq2Seq]: what do you do for a living ?
Enter Your Message: I work as a trader
[Seq2Seq]: what city are you from ?
Enter Your Message: delhi india
[Seq2Seq]: i am in cali
Enter Your Message: how is it like there
[Seq2Seq]: it is beautiful there are a lot of people out there
Enter Your Message: that's nice
[Seq2Seq]: it really is that's why i moved here
Enter Your Message: where were you previously
[Seq2Seq]: i was previously born and raised
Enter Your Message: where
[Seq2Seq]: where did you move to
Enter Your Message: I move to delhi recently
[Seq2Seq]: i grew up in india
Enter Your Message: me too
[Seq2Seq]: i do not know what i want to do yet
Enter Your Message: you will
[Seq2Seq]: i will have to see if i can find it
Enter Your Message:
```

Figure 5.7: Conversation Snippet for the model with Response Relatedness control using WD and repetition control using WD.

Sometimes the responses from the model are incomplete and that makes no sense at all. This leads to the overall poor quality of the conversational model.

5.8 conclusion & Future Work

Developing a fluent open-ended dialog agent is a hard task, a task where not only a large model with lots of weights matter but also how we are gonna evaluate in real life matters too. This chapter makes us realize just how important the decoding part of the neural conversational models matters. This chapter also provides a guide as to what really makes a good conversation. As from our observations, we infer that repetition removal is one such pivotal attribute that should always be implemented through decoding or conditional training in all of our neural conversation model. This model provides a much-needed proof that including control in our models always makes them better if correctly fed.

Future work can include as we noted from our observation as to how we can model consistency of the conversational model, factual attribute, or the ability of the dialogue systems to give correct answers to factual queries. Future work can also include how we can train our models in order to make them more knowledgeable to get common-sense knowledge right in their responses.

Conclusion 6

The ability to converse with a human fluently and naturally is one of the most complex tasks for the machines. The work presented here in the dissertation provides an unexhaustive survey of various techniques and methods used for building such dialog systems. These methods and techniques are the culmination of decades of research in Machine Learning, Deep Learning, Linguistics, and Computer Science.

In chapter 3, we discussed various deep learning models that are employed in the tasks of Natural Language Generation. We mainly saw two of the most popular architectures - Recurrent Neural Network-based and Transformer based involved in the task of open-ended conversations. We found out that the Transformer model outperforms the RNN model by a large margin and requires less parameter than the RNN model. We also learned the fact that the algorithm involved in decoding or generating a text is equally important. The most important learning from the whole chapter is "a clever algorithm/model can do wonders but the path to one involves time and patience".

In chapter 4, we discussed the different methods for including the context comprising of multiple dialogues in our dialogue systems. We also discuss how explicit traits like knowledge, persona, etc can be included alongside the context in our dialogue systems. Although, we cannot converse with a knowledge-based conversational agent, from the snippets from the respective paper we analyzed, including these always increases the quality of responses.

In chapter 5, we discussed the methodologies of modeling some of the attributes like response relatedness, question-asking, etc that we found missing in our conversations in chapter 3. We qualitatively study how the inclusion of control in our model can lead to better conversations. We also briefly discuss the attributes and parameters that make a conversation good.

In all, we discussed the important ingredients needed to make an engaging and interesting dialogue system. We discussed the data-sets required, the models required and the generation algorithms needed to build such systems.

6.1 Future Work

Most of the techniques and methods discussed in this dissertation are not older than a decade. Some of these methods like Nucleus Sampling [44] has recently been introduced in ICLR 2020[International Conference Learning Representations 27-30 April 2020]. The field is rapidly evolving as a lot of people around the world both in academia and industries are working hard to become the leader. For example, In January 2020, Google released Meena Bot [52] for which they claimed that it performs better than any of the open-ended conversational agent in existence, just a day earlier[as of writing this dissertation] Facebook released Blender [53] for which they claimed that it outperformed Meena bot. These systems are trained on very large data and consists of billions of parameters for a comparison look at Table 6.1. Training them requires a great deal of computation power for example according to this, Meena bot required over 10 Crore Rupees of computing time to become as good as the claim. The mammoth size of these models hinders them to be used on an ordinary computer system. So, if we can't scale these models, we don't have use for these.

Model	Parameter Count
Meena Bot	2.4 Billion
Blender Bot	9.4 Billion
Seq2seq Transformer [Chapter 3]	8.6 Million

Table 6.1: Comparison of our model with state of the art dialogue systems.

We also require our system to have consistency when talking with someone. Here consistency refers to not contradicting itself during the course of conversation. From our conversing experience with various systems, we also would like to have a factual system, the one which gets factual information right.

If the above efforts are successful, then we can have much smarter machines which will eventually be more functional and would require less human effort. We think such systems will in due course reduce human labor to a considerable level and will surely help us in sectors like hospitals, which is the need for the hour. We also believe development in this field will lead to a significant breakthrough in General Artificial Intelligence.

Bibliography

Here are the references in citation order.

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. Ed. by F. Pereira et al. 2012. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf (cited on page 1).

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2014 (cited on pages 1, 4, 8, 18, 25, 27).

[3] Ashish Vaswani et al. *Attention is All you Need*. Ed. by I. Guyon et al. 2017. URL: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf (cited on pages 1, 5, 8, 17, 18, 25–27).

[4] David Silver et al. 'Mastering the game of Go without human knowledge'. In: *Nature* 550 (Oct. 2017), pp. 354– (cited on page 1).

[5] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015 (cited on pages 1, 22).

[6] Christian Szegedy et al. *Rethinking the Inception Architecture for Computer Vision*. 2015 (cited on page 1).

[7] Mingxing Tan et al. *MnasNet: Platform-Aware Neural Architecture Search for Mobile*. 2018 (cited on page 1).

[8] Jianfeng Gao, Michel Galley, and Lihong Li. *Neural Approaches to Conversational AI*. 2018 (cited on page 2).

[9] Tsung-Hsien Wen et al. *A Network-based End-to-End Trainable Task-oriented Dialogue System*. 2016 (cited on page 2).

[10] Oriol Vinyals and Quoc Le. *A Neural Conversational Model*. 2015 (cited on page 2).

[11] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*. 2014 (cited on pages 2, 3, 7, 18, 27).

[12] Junyoung Chung et al. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 2014 (cited on page 3).

[13] Sepp Hochreiter and Jürgen Schmidhuber. 'Long Short-Term Memory'. In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735 (cited on pages 4, 22).

[14] Kyunghyun Cho et al. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014 (cited on pages 4, 17, 23).

[15] Tomas Mikolov et al. *Distributed Representations of Words and Phrases and their Compositionality*. Ed. by C. J. C. Burges et al. 2013. URL: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf (cited on page 7).

[16] Yoshua Bengio et al. 'A Neural Probabilistic Language Model'. In: *J. Mach. Learn. Res.* 3.null (Mar. 2003), pp. 1137–1155 (cited on page 7).

[17] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. *GloVe: Global Vectors for Word Representation*. 2014. URL: http://www.aclweb.org/anthology/D14-1162 (cited on page 7).

[18] Matthew E. Peters et al. *Deep contextualized word representations*. 2018 (cited on page 7).

[19] Niki Parmar et al. *Image Transformer*. 2018 (cited on page 8).

[20] Cheng-Zhi Anna Huang et al. *Music Transformer*. 2018 (cited on page 8).

[21] Alessandro Sordoni et al. *A Hierarchical Recurrent Encoder-Decoder For Generative Context-Aware Query Suggestion*. 2015 (cited on pages 8, 44).

[22] Saizheng Zhang et al. *Personalizing Dialogue Agents: I have a dog, do you have pets too?* 2018 (cited on pages 8, 51).

[23] Emily Dinan et al. *Wizard of Wikipedia: Knowledge-Powered Conversational agents*. 2018 (cited on pages 8, 46, 47).

[24] Yuta Kikuchi et al. *Controlling Output Length in Neural Encoder-Decoders*. Austin, Texas, Nov. 2016. DOI: 10.18653/v1/D16-1140. URL: https://www.aclweb.org/anthology/D16-1140 (cited on pages 8, 50).

[25] Nanyun Peng et al. *Towards Controllable Story Generation*. New Orleans, Louisiana, June 2018. DOI: 10.18653/v1/W18-1505. URL: https://www.aclweb.org/anthology/W18-1505 (cited on pages 8, 50).

[26] Angela Fan, Mike Lewis, and Yann Dauphin. *Hierarchical Neural Story Generation*. 2018 (cited on pages 8, 29, 30, 50).

[27] Marjan Ghazvininejad et al. *Hafez: an Interactive Poetry Generation System*. Vancouver, Canada, July 2017. URL: https://www.aclweb.org/anthology/P17-4008 (cited on page 8).

[28] Ashutosh Baheti et al. *Generating More Interesting Responses in Neural Conversation Models with Distributional Constraints*. Brussels, Belgium, Oct. 2018. DOI: 10.18653/v1/D18-1431. URL: https://www.aclweb.org/anthology/D18-1431 (cited on pages 8, 51).

[29] Abigail See et al. *What makes a good conversation? How controllable attributes affect human judgments*. 2019 (cited on pages 8, 50, 51).

[30] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006 (cited on pages 10, 13, 17).

[31] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cited on pages 10–14, 17, 21, 27).

[32] Aston Zhang et al. *Dive into Deep Learning*. https://d2l.ai. 2020 (cited on pages 10, 17).

[33] Matthew D Zeiler and Rob Fergus. *Visualizing and Understanding Convolutional Networks*. 2013 (cited on page 14).

[34] Jiwei Li et al. *A Persona-Based Neural Conversation Model*. Berlin, Germany, Aug. 2016. DOI: 10.18653/v1/P16-1094. URL: https://www.aclweb.org/anthology/P16-1094 (cited on page 15).

[35] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning Representations by Back-Propagating Errors*. Cambridge, MA, USA, 1988 (cited on page 16).

[36] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019 (cited on page 16).

[37] Ilya Sutskever et al. *On the Importance of Initialization and Momentum in Deep Learning*. Atlanta, GA, USA, 2013 (cited on page 16).

[38] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014 (cited on pages 16, 32).

[39] Yann LeCun et al. *Gradient-Based Learning Applied to Document Recognition*. 1998. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.7665 (cited on page 16).

[40] Alec Radford et al. 'Language Models are Unsupervised Multitask Learners'. In: (2018) (cited on page 17).

[41] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. *Effective Approaches to Attention-based Neural Machine Translation*. 2015 (cited on pages 18, 25, 27).

[42] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. *On the Difficulty of Training Recurrent Neural Networks*. Atlanta, GA, USA, 2013 (cited on page 21).

[43] Ari Holtzman et al. *Learning to Write with Cooperative Discriminators*. Melbourne, Australia, July 2018. DOI: 10.18653/v1/P18-1152. URL: https://www.aclweb.org/anthology/P18-1152 (cited on page 29).

[44] Ari Holtzman et al. *The Curious Case of Neural Text Degeneration*. 2019 (cited on pages 29, 30, 57).

[45] Alec Radford. *Improving Language Understanding by Generative Pre-Training*. 2018 (cited on page 30).

[46] Massimo Caccia et al. *Language GANs Falling Short*. 2018 (cited on page 30).

[47] Tatsunori B. Hashimoto, Hugh Zhang, and Percy Liang. *Unifying Human and Statistical Evaluation for Natural Language Generation*. 2019 (cited on page 30).

[48] Cristian Danescu-Niculescu-Mizil and Lillian Lee. *Chameleons in imagined conversations: A new approach to understanding coordination of linguistic style in dialogs.* 2011 (cited on page 30).

[49] Iulian V. Serban et al. *Building End-To-End Dialogue Systems Using Generative Hierarchical Neural Network Models*. 2015 (cited on pages 44, 45).

[50] Yizhe Zhang et al. *DialoGPT: Large-Scale Generative Pre-training for Conversational Response Generation*. 2019 (cited on page 51).

[51] A. H. Miller et al. 'ParlAI: A Dialog Research Software Platform'. In: *arXiv preprint arXiv:1705.06476* (2017) (cited on page 51).

[52] Daniel Adiwardana et al. *Towards a Human-like Open-Domain Chatbot*. 2020 (cited on page 57).

[53] Stephen Roller et al. *Recipes for building an open-domain chatbot*. 2020 (cited on page 58).