YILDIZ TECHNICAL UNIVERSITY
FACULTY OF CHEMISTRY AND METALLURGY
MATHEMATICAL ENGINEERING

Design Applications in Mathematical Engineering:

Question Generation for Reading Compherension using Deep Learning

---

Tuğrul Hasan Karabulut, 17058055
Thesis Advisor: Prof. İbrahim Emiroğlu

Istanbul, 2020

Contents

List of Figures iv

List of Tables v

List of Symbols vi

List of Abbreviations vii

1 Introduction 1

2 Deep Learning 1

2.1 Machine Learning 1

2.1.1 Types of Learning 1

2.1.2 Experimental Setup 2

2.2 Some Machine Learning Models 2

2.2.1 Linear Regression 2

2.2.2 Logistic Regression 3

2.3 Training Procedure 4

2.3.1 Loss Functions 4

2.3.2 Optimization 5

2.4 Deep Learning 7

2.4.1 Activation Functions 8

2.5 Backpropagation 10

2.6 Optimization Algorithms in Deep Learning 10

2.6.1 Stochastic Gradient Descent 11
2.6.2 Adam: A Method For Stochastic Optimization 11
2.7 Regularization 12
2.8 Sequence Modelling 13
2.9 Types of RNNs 13
2.9.1 Vanilla RNN 13
2.9.2 LSTM 14
2.9.3 GRU 16
2.10 Encoder-Decoder Architecture 16
2.11 Attention 17
2.11.1 Bahdanau Attention 18

3 Natural Language Processing **19**

3.1 NLP Tasks 19
3.2 NLP and Deep Learning 19
3.3 Word Representations 19
3.3.1 Word2Vec 20
3.3.2 GloVe 20

4 Project **22**

4.1 Question Generation 22
4.2 Related Work 22
4.3 Data Set 22
4.4 Data Analysis and Preprocessing 23

4.5 Pretrained Word Embeddings 25
4.6 Model 25
4.6.1 Training 25
4.7 Inference 27
4.8 Evaluation 28
4.8.1 BLEU 28
4.8.2 ROUGE 29
4.8.3 Results 29
4.9 Software Tools 29
4.10 Other Parts of the Project 29
4.10.1 Text Summarization 30
4.10.2 Mobile Application Development 30

5 Conclusion and Future Work **31**

References **32**

List of Figures

2.1 Sigmoid 3

2.2 A Shallow Neural Network 7

2.3 ReLU 9

2.4 tanh 10

2.5 Recurrent Neural Network 13

2.6 LSTM Network 15

2.7 GRU Network 16

2.8 Encoder Decoder 17

4.1 Distribution of lengths of the input texts 24

4.2 Distribution of lengths of the question texts 24

4.3 Validation loss across epochs of the model without answers 26

4.4 Validation loss across epochs of the model with answers 26

4.5 Beam Search Example (with beam width = 5) 27

List of Tables

4.1 Metrics on the test set 29

List of Symbols

x	scalar
$\mathbf{x}$	vector
X	set
$\mathbf{X}$	matrix

List of Abbreviations

RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
BLEU	Bilingual Evaluation Understudy
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
Adam	Adaptive Moment Estimation
NLP	Natural Language Processing

Abstract

In today's world, lots of tasks that once required human labor are now being automated by deep learning models. Self-driving cars, translators, chatbots and many more products are being shipped into production. In this project, a way to automate question generation from a text passage is investigated. To do that, we need a model that takes a text sequence that contains information and returns another text sequence which is a question that is related to the input text. It turns out that deep learning is heavily used to do such tasks. In this study, we explore deep learning and natural language processing and how to use these to generate questions from text.

1 Introduction

This project is actually a part of another project that is done in collaboration with Tuğrul Hasan Karabulut, Ayşe Duman and Eda Nur Ersu. This project is about making a mobile application that displays summaries of long informative texts to the users and testing their understanding of the subject by asking them some generated questions. Both summaries and questions is automatically generated using deep learning models. Ayşe Duman was responsible for generating summaries from text sequences and Eda Nur was responsible for the development of mobile application.

2 Deep Learning

Deep learning is originally a sub-branch of machine learning. Machine learning is basically a set of methods that create predictive models using some sample data and statistics. These models need to be implemented in software and trained. For training, mathematical optimization techniques are used. In this chapter, we will see, starting with Machine Learning, how Deep Learning works and which type of models are commonly used.

2.1 Machine Learning

Machine learning is programming computers by showing them some example data to improve a performance criterion or to solve a problem [1]. It is an interdisciplinary field based on statistics, mathematical optimization and computer science.

2.1.1 Types of Learning

In machine learning, we usually start with obtaining a data set that is relevant to the problem that we want to solve. A data set has observations taken from various sources. An observation has a set of features that describe itself. If our task involves predicting an output based on an input, then this is called *supervised learning*. The output we want to predict is called *target feature*, others are called *input features*. Target feature can be categorical or continuous. If it is categorical, then our task is *classification*. For example, given the information of a customer, predict whether she/he will continue buying services in the next month or not. If the target feature is continuous, then our task is *regression*. For example, predicting house prices, currencies etc.

In a lot of cases, we may not want to predict a target feature but rather, we may want to

find some structure or pattern in our data set. For example, we can build some learning model that clusters our data set into different clusters based on the similarities of observations. Type of Machine Learning that deals with these cases is *unsupervised learning*. Common unsupervised learning examples are market segmentation and image compression.

2.1.2 Experimental Setup

Before building a Machine Learning model, we usually split our data set into three pieces: training set, validation set and test set. Training set, as the name suggests, is used for training the model. Validation set is used to tune the parameters of the model that needs to be evaluated beforehand. These type of parameters are called *hyperparameters*. By tuning, we mean finding the optimum (or at least approriate) values. We are tuning the hyperparameters by doing experiments on the training set with different hyperparameters and evaluating the model's performance on the validation set. After finding the best hyperparameters and successfully training the model, we evaulate the model on a completely unseen data set besides training and validation sets. For this, we use the test set. Test set performance is the most reliable evidence for the success of the model and is usually the one that's reported on research.

2.2 Some Machine Learning Models

Now, let us see some examples of machine learning models. We will talk about two basic models. One of them is for regression which is linear regression. It is the simplest model for regression tasks. Other one is the logistic regression which is used for classification. They are actually more of a statistical model. Both models are part of the family of models called *generalized linear models*.

2.2.1 Linear Regression

Suppose that we have a training set $X = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)})\}$. We want to predict $y^{(i)}$, the output variable, based on the input features, $\mathbf{x}^{(i)}$. $\mathbf{x}^{(i)}$ is an n-dimensional feature vector. Using this training data, we want to find a function $f(\mathbf{x})$ that predicts the output variable y. y is a continuous value. We model these problem as a parametric function, $f(\mathbf{x};\theta)$, which is:

$$f(\mathbf{x};\theta) = \theta^T \mathbf{x} = \mathbf{x}_0\theta_0 + \mathbf{x}_1\theta_1 + \cdots + \mathbf{x}_n\theta_n \tag{2.1}$$

which is the *linear regression* model.

2.2.2 Logistic Regression

What if our task is classification, such as classifying emails as spam or not? Then, we can use logistic regression model. We need only to make a slight modification to the linear regression model. Linear regression model outputs any real number. But for classification, we need probabilities. For that, we use *sigmoid* function:

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{2.2}$$

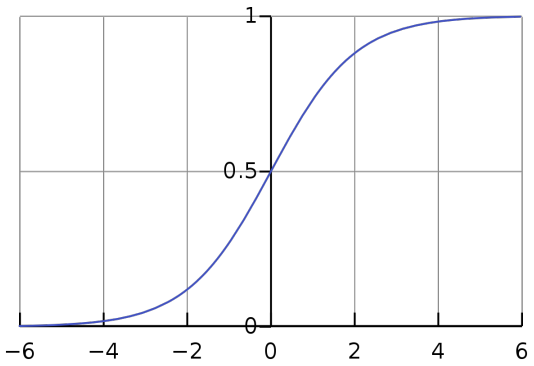

Figure 2.1: Sigmoid
[2]

Sigmoid function outputs a positive number if $z >= 0$, and a negative number otherwise. To turn this into a parametric model in order to learn the optimum fit to the data, we use sigmoid function with our $f(\mathbf{x};\theta) = \theta^T\mathbf{x}$ as its input, that is:

$$g(\mathbf{x};\theta) = \frac{1}{1+e^{-\theta^T\mathbf{x}}} \tag{2.3}$$

which is the *logistic regression* model.

2.3 Training Procedure

In the previous section, we saw two models: linear regression and logistic regression. These are parametric models. Their parameters needs to be optimized so that our model produces accurate results. We need to define two things before training our model: optimization criterion and optimization algorithm.

2.3.1 Loss Functions

In optimization, loss function is a function that indicates the error of the model. We expect an high loss value if the model's output to an observation and the actual output is dramatically different and vice versa.

Loss functions differ according to different tasks and data. If we want to predict a continuous value, for example house prices, then common loss functions include: *squared error*, *absolute error*.

Terminology Loss function and cost function used interchangeably but usually, loss function refers to the loss of a single training example while cost function refers to the average loss of whole training set.

Mean squared error (MSE) is the cost function that uses squared error:

$$\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}-\hat{y}^{(i)})^2 \tag{2.4}$$

So, we penalize the model according to the square of the difference between the true value and the predicted value.

Mean absolute error(MAE) is the cost function that uses absolute error:

$$\frac{1}{m}\sum_{i=1}^{m}|y^{(i)}-\hat{y}^{(i)}| \tag{2.5}$$

Mean absolute error is usually used to deal with outlier values in the training set, as the model may not be able to produce good results for outlier values. But, mean square error is better for helping the model to correct small errors. So, both loss

functions have their own advantages. There is another loss function that combines squared error and absolute error which is *Huber loss*:

$$\begin{cases} (y^{(i)} - \hat{y}^{(i)})^2 & \text{if } |y^{(i)} - \hat{y}^{(i)}| \leq \delta \\ \delta |y^{(i)} - \hat{y}^{(i)}| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases} \tag{2.6}$$

If we want to predict a discrete value, that is, if we want to do classification, then *cross entropy loss* is used:

$$-\sum_{i=1}^{K} y_i \log(\hat{y}_i) \tag{2.7}$$

where K is the number of distinct values, or classes and $\hat{y}$ is the predicted probability for the observation y.

2.3.2 Optimization

Training of a machine learning model is done by optimizing the parameters by trying to minimize a pre-defined loss function. Numerical optimization algorithms is usually used. In most applications, *gradient descent* or a variant of gradient descent algorithm is used to optimize the parameters of the model. Gradient descent works as incrementally updating the parameters of the model by subtracting the partial derivative of the cost function with respect to the corresponding parameter.

Suppose that we have a cost function $H(\Theta)$ where Θ is the parameters of our model. In gradient descent we update our parameters as follows:

$$\Theta_i := \Theta_i - \alpha \frac{\partial H(\Theta)}{\partial \Theta_i} \tag{2.8}$$

where α is an hyperparameter that defines the step size for a parameter update. It is called *learning rate*.

We can also write the whole update equation by using vector calculus notation:

$$\Theta := \Theta - \alpha \nabla_{\Theta} H(\Theta) \tag{2.9}$$

where $\nabla_\Theta H(\Theta)$ is the gradient of the cost function with respect to the parameter vector Θ.

Let us now see an example of gradient descent in linear regression. Suppose we use mean squared error as our cost function:

$$H(\Theta) = \frac{1}{2m} \sum_{i=1}^{m} (y^{(i)} - f(\mathbf{x}^{(i)}; \Theta))^2 \tag{2.10}$$

Partial derivative of $H(\Theta)$ with respect to parameter Θ_i can be written as:

$$\frac{\partial H(\Theta)}{\partial \Theta_i} = \frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - f(\mathbf{x}^{(i)}; \Theta)) \mathbf{x}_i^{(i)} \tag{2.11}$$

Therefore, gradient descent update equation for linear regression can be written as:

$$\Theta_i := \Theta_i - \alpha \frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - f(\mathbf{x}^{(i)}; \Theta)) \mathbf{x}_i^{(i)} \tag{2.12}$$

Learning Rate α is a very important hyperparameter that needs to be tuned. If α is high, then gradient descent takes big step towards the minimum point and it might miss the optimum point. if α is small, then gradient descent takes very small steps towards the minimum point and the convergence of the algorithm might take so much time. That's why, we need to find a good value for learning rate.

Basic optimization process in machine learning is discussed in this section. In deep learning, much more sophisticated optimization algorithms is used. Optimizing deep learning models is a non-convex problem and it needs more attention than simple convex models. We will discuss training deep learning models in the upcoming sections.

2.4 Deep Learning

Deep learning is based on the model called *artificial neural network*. Neural networks consist of neurons that propagate signals to each other. Neural networks are somewhat inspired by biological neural networks in the brain. [3]

A neural network consists of multiple layers. First, there is an input layer that has our input features of our data set. After the input layer, there are hidden layers where each hidden layer has a group of neurons (or hidden units) that computes activation values and propagate it to the next layer. Finally, there is the output layer that outputs the result of the computation in the neural network. If a neural network has only one hidden layer, then it is called a *shallow neural network*. If it has multiple hidden layers, then it is a *deep neural network*. A basic, shallow neural network architecture is given in Figure 2.2

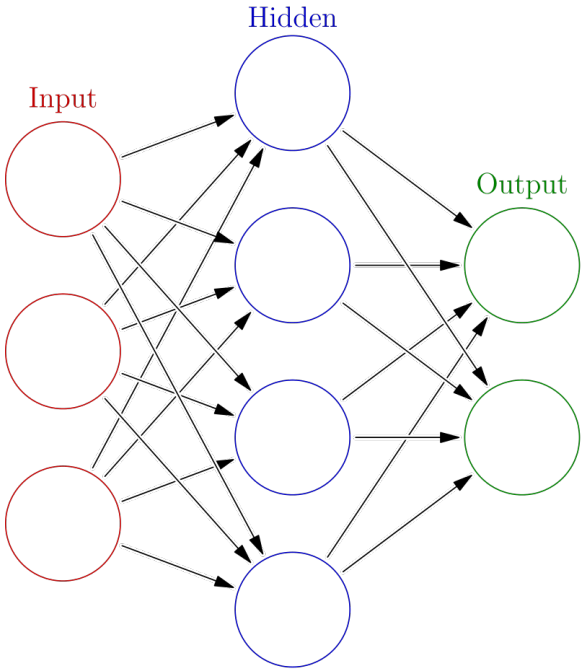

Figure 2.2: A Shallow Neural Network
[4]

If you recall from the previous sections, we discussed linear regression and logistic regression models. Both of these models use a feature vector, that is our input $\mathbf{x}$ and we have a parameter vector Θ. Every feature has a corresponding parameter. $\mathbf{x}$ consists of all the features we use and we extract these features by analyzing our raw data set. Neural network model has a different approach that makes it a well-performing model. Advantage of a neural network is that it takes input features and creates composed, higher level representation of features in the hidden layers. Every

hidden layer creates its own features by composing the previous layer's features. It does this by using weights between the layers and optimizing these weights. $W^{[l]}$ is a weight matrix that belongs to the layer l of a neural network. It is of size $n^{[l]} \times n^{[l-1]}$, where $n^{[l]}$ is the number of neurons in the lth and $n^{[l-1]}$ is the number of neurons in the $l-1$th layer.

In every layer of a neural network, there are two steps of computation. First is computing a linear expression using the weights and giving this result to a function called *activation function*. An activation function is usually a non-linear function and is used to learn complex relationships in the data. Common activation functions include: ReLU (rectified linear unit), sigmoid, tanh. Below is the forward computation in a hidden layer of a neural network.

$$\mathbf{z}^{[l]} = \mathbf{W}^{[l]}\mathbf{a}^{(l-1)} + \mathbf{b}^{[l]} \tag{2.13}$$

$$\mathbf{a}^{[l]} = g(\mathbf{z}^{[l]}) \tag{2.14}$$

where $g(\mathbf{z})$ is an activation function.

$\mathbf{W}^{[l]}$ and $\mathbf{b}^{[l]}$ are parameters of the lth layer. After the forward propagation, we do *backpropagation* to update these parameters. Backpropagation is a method for effectively computing the gradients of the cost function with respect to the model parameters.

2.4.1 Activation Functions

Activation functions are important part of neural networks. They help the neural network to learn non-linear functions. Most of the real world tasks are non-linear problems. Therefore, activation functions are used in the neural networks.

Sigmoid, tanh and ReLU are common activation functions. In today's applications, ReLU (Rectified Linear Unit) is the recommended activation function to use in the hidden layer. ReLU helps to prevent a problem called *vanishing gradient*. It maps the input to the positive part of its input. Formally, it can be defined as follows:

$$\text{ReLU(z)} = \max(0, z) \tag{2.15}$$

Although it seems as a linear function, ReLU is effective for learning non-linear functions because it is not really a linear function. It is a combination of two linear functions $f(x) = 0$ and $f(x) = z$. Its discontiunity makes it a non-linear function. Also,

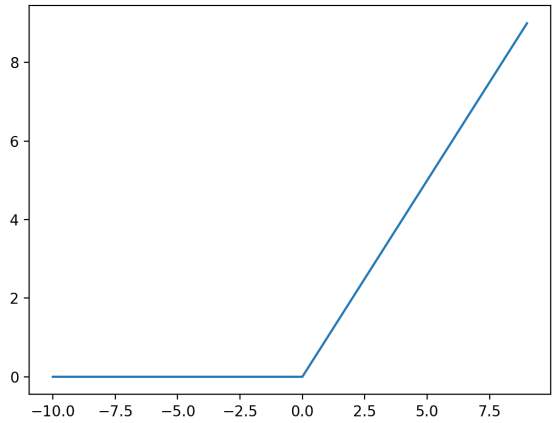

Figure 2.3: ReLU

its gradients' magnitude is 1 for positive side of z. This is helpful in the optimization process of neural networks because we usually don't want gradients to be close to zero or too big.

Another activation function is sigmoid. As we discussed, it is used in logistic regression. It maps its input to a value that is between 0 and 1. In another words, it maps its input to a probability. It was usually used as activation function in the hidden layers before ReLU is found be effective in neural networks.

tanh is another activation that is commonly used. It is also a non-linear function. It is usually used in recurrent neural networks (RNN). tanh maps its input to a value between -1 and 1.

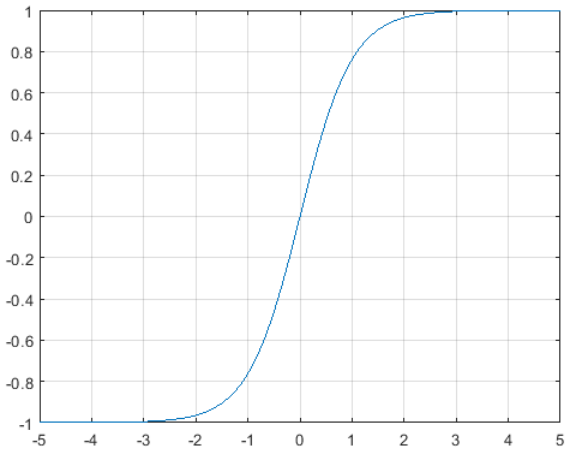

Figure 2.4: tanh

2.5 Backpropagation

Backpropagation is an algorithm to compute the gradients of the weights in a neural network. These gradients are calculated in a way that any redundant or repeated calculation is avoided. Gradients are calculated using the chain rule in calculus.

Derivatives are computed with respect to the cost function. Starting from last layer, L, gradient of every parameter is calculated and stored in order to avoid repetitive computation. This process is continued until we reach to the first layer.

After the computation of the gradients, parameters are updated by an optimization algorithm such as gradient descent, stochastic gradient descent, etc.

2.6 Optimization Algorithms in Deep Learning

There are many optimization algorithms that are developed for training deep neural networks. Usually, neural networks have millions or billions of parameters. Calculations of their gradients and updating them are expensive calculations. So, we want to train our networks by doing these steps as few as possible. But, on the other hand, we want to reduce the time of updating the parameters because we want to see the improvement of the network.

2.6.1 Stochastic Gradient Descent

In stochastic gradient descent (SGD), instead of updating the parameter using the whole training set, we use only a small subset of the training set. For this, we define an hyperparameter, batch size, which denotes how many examples we use to do a single update. Batch size is usually chosen to be between 1 and few hundred. This way, we see the improvement of the model instead of waiting for the algorithm to process the whole training data, which is in practice, infeasible.

If the algorithm once go over the whole training data batch by batch, we call that an *epoch*. We determine our training time by number of epochs which is also an hyperparameter. Equation below shows the gradient calculation over a batch.

$$\mathbf{g} = \frac{1}{m'} \Delta_\theta \sum_{i=1}^{m'} L(\mathbf{x}_i, y_i, \theta) \tag{2.16}$$

2.6.2 Adam: A Method For Stochastic Optimization

Problem of SGD is that different batches may differ and gradient computation may be noisy. This can prevent the algorithm from converging and/or increase the training time. Adam (stands for Adaptive Moment Estimate) helps to reduce this problem by using exponentially moving averages to smooth the first and second moments of gradients across time. Nowadays, Adam is the go-to optimization algorithm for many tasks. Computations of Adam are given below.

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t \tag{2.17}$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)\mathbf{g}_t^2 \tag{2.18}$$

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1-\beta_1^t} \tag{2.19}$$

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-\beta_2^t} \tag{2.20}$$

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon} \tag{2.21}$$

where g_t is the gradient at time t. β_1, β_2, ϵ and α are hyperparameters. β's values are usually close to 1. ϵ is to prevent from division by zero.

2.7 Regularization

In machine learning problems, there may be a problem that the model performs well on the training set but gives poor performance on unseen data. Reason for this is that models, instead of generalizing using the training data, it overlearns (or memorizes) the training set, This problem is called *overfitting*. *Regularization* techniques try to overcome this problem. One popular regularization technique is L_2 regularization. L_2 regularization adds a term to the loss function. This term is a summation over the square of the parameters in the model. By adding this term, we prevent the norm of the parameters from getting too big. Equations below is a regularized cost function of linear regression. λ is the regularization parameter. It defines the regularization strength. It can be increased to increase the regularization of the model. Regularized linear regression cost function is given below:

$$\sum_{i=1}^{N}(y_i - f(\mathbf{x}_i;\theta))^2 + \sum_{i=1}^{n}\lambda\theta_i^2 \tag{2.22}$$

One other regularization technique which is commonly used in deep learning is *dropout*. Dropout works by randomly dropping out the outputs of some neurons in a hidden layer. By dropping out, we mean that we set them equal to zero. This prevents the model from getting too dependent on specific neurons and thus reduces overfitting. Ratio of the neurons that will be dropped out is determined beforehand.

2.8 Sequence Modelling

The neural network architecture presented above works well for non-sequential data. There is another network architecture for working on sequence data such as text, time series, etc. This types of networks are called *recurrent neural networks*.

Recurrent neural network (RNN) is a special kind of neural network where there are connections between the nodes in the layer. A node use previous node's internal state to produce its output and its internal state and it passes into the next node. It processes a sequence, $\mathbf{x}$, one by one. A recurrent neural network architecture is given in Figure 2.5

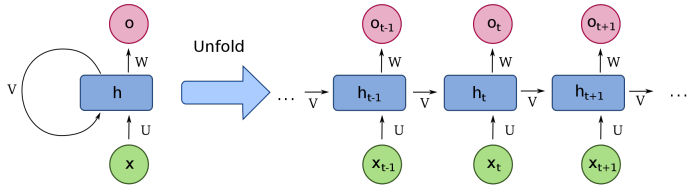

Figure 2.5: Recurrent Neural Network
[5]

2.9 Types of RNNs

In RNNs, directions can also go both ways. This results in *bidirectional RNN*. This types of RNNs process sequences both left-to-right and right-to-left. This helps learning the dependencies between the parts of the sequence better but it also slows down the training process as it does the sequential process step two times.

There is an RNN architecture which is popularly used: *LSTM (Long short-term memory)*. It allows us to easily learn the long term relationships in the sequence.

Now, let us explore different types of RNNs.

2.9.1 Vanilla RNN

Suppose that $\mathbf{x}^{(t)}$ represents the input at timestep t and $\tilde{\mathbf{y}}^{(t)}$ represents the output at timestep t.

A vanilla RNN layer does the following computations:

$$\boldsymbol{\alpha}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} \tag{2.23}$$

$$\mathbf{h}^{(t)} = \tanh(\boldsymbol{\alpha}^{(t)}) \tag{2.24}$$

$$\mathbf{z}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)} \tag{2.25}$$

$$\tilde{\mathbf{y}}^{(t)} = \text{softmax}(\mathbf{z}^{(t)}) \tag{2.26}$$

Let us make sense of these computations. $\mathbf{h}^{(t)}$ is the hidden state of the RNN layer at timestep t. tanh activation function is used to calculate the hidden state. **W**, **U** and **b** are parameters that are learned by the network to calculate accurate hidden states of the layers. These parameters are shared across all the layers in order to build a generalized model [6]. $\tilde{\mathbf{y}}^{(t)}$ is the output of the recurrent layer. It is calculated using softmax activation function. **c** and **V** are also parameters that are learned by the network to calculate the correct output.

In practice, Vanilla RNNs have the problem of learning long term dependencies. Also, exploding and vanishing gradients badly affect the training process. LSTM and GRU architectures solve this problem.

2.9.2 LSTM

LSTM stands for Long short-term memory. It is a special kind of RNN that is capable of learning long term dependencies. It has been proposed by Hochreiter & Schmidhuber (1997) [7]. It is widely used in problems where there are sequential data such as text, time series, etc.

An important aspect of LSTMs is that they have two internal states: hidden state and cell state. Cell states is the combination of output of two gates that are present in LSTM. These gates are forget gate and input gate.

First step in an LSTM is to calculate the cell state. We first feed the previous hidden state and input to the forget gate to decide which information we will forget and which information we will keep. After that, we again use the previous hidden state and input to decide we will store in the cell state. We use the input gate to carry out this computation. Forget uses the sigmoid activation function while the input gate uses the tanh activation function. As in vanilla RNN, we need to calculate the output and hidden state of the layer. In mathematical form, these computations are the following equations.

$$\mathbf{f}^{(t)} = \sigma(\mathbf{W}_f[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_f) \tag{2.27}$$

$$\mathbf{i}^{(t)} = \sigma(\mathbf{W}_i[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_i) \tag{2.28}$$

$$\tilde{\mathbf{C}}^{(t)} = \tanh(\mathbf{W}_C[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_C) \tag{2.29}$$

$$\mathbf{C}^{(t)} = \mathbf{f}^{(t)} * \mathbf{C}^{(t-1)} + \mathbf{i}^{(t)} * \tilde{\mathbf{C}}^{(t)} \tag{2.30}$$

$$\mathbf{o}^{(t)} = \sigma(\mathbf{W}_o[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_o) \tag{2.31}$$

$$\mathbf{h}^{(t)} = \mathbf{o}^{(t)} * \tanh(\mathbf{C}^{(t)}) \tag{2.32}$$

where $\mathbf{C}^{(t)}$ is the cell state at timestep t, $\mathbf{o}^{(t)}$ is the output at timestemp t and $\mathbf{h}^{(t)}$ is the hidden state at timestep t.

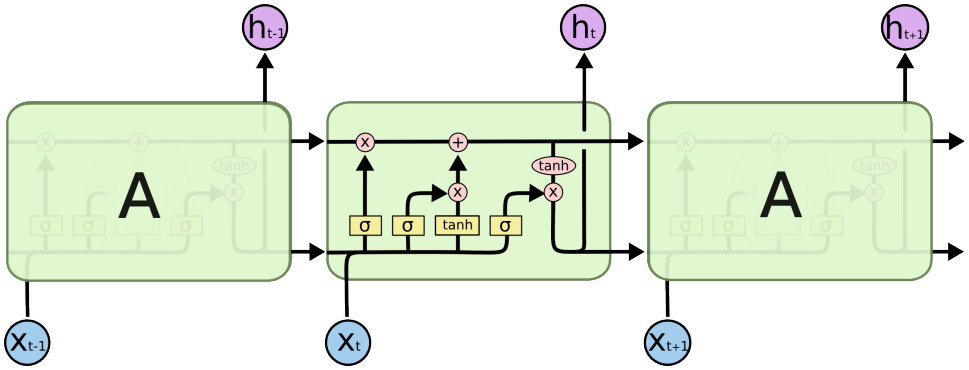

Figure 2.6: LSTM Network
[8]

2.9.3 GRU

GRU is another RNN architecture that has been proposed by Cho, et al. (2014). GRU is similar to LSTM except that it does not have an output gate like LSTM. It has update and reset gates. Therefore, it has fewer parameters and this makes it faster to train. Also, it has only one state. GRU equations are given below:

$$\mathbf{i}^{(t)} = \sigma(\mathbf{W}_i[\mathbf{C}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_i) \tag{2.33}$$

$$\mathbf{r}^{(t)} = \sigma(\mathbf{W}_r[\mathbf{C}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_f) \tag{2.34}$$

$$\tilde{\mathbf{C}}^{(t)} = \tanh(\mathbf{W}_C[\mathbf{r}^{(t)} * \mathbf{C}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_C) \tag{2.35}$$

$$\mathbf{C}^{(t)} = (1 - \mathbf{i}^{(t)}) * \mathbf{C}^{(t-1)} + \mathbf{i}^{(t)} * \tilde{\mathbf{C}}^{(t)} \tag{2.36}$$

$$\mathbf{h}^{(t)} = \mathbf{C}^{(t)} \tag{2.37}$$

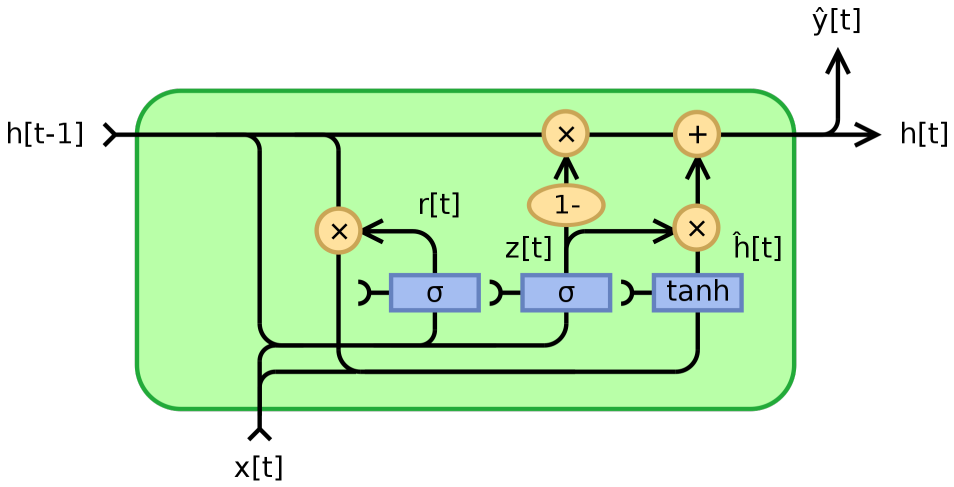

Figure 2.7: GRU Network
[9]

2.10 Encoder-Decoder Architecture

Encoder-Decoder Recurrent Neural Network architectures are widely used to solve tasks that require sequence-to-sequence learning. These tasks are usually as follows: given an input sequence $\mathbf{x}$, output another sequence $\mathbf{y}$, relevant to the training set $\{\mathbf{X}, \mathbf{Y}\}$. Common sequence-to-sequence learning tasks are machine translation, text summarization, chatbot and image captioning.

Let us now explore how encoder-decoder models work. In an encoder-decoder model, we have two separate components: encoder and decoder. Encoder creates a dense

and lower dimensional representation of the input data. For example, suppose that an input text with m words with its n dimensional embeddings given. This makes the input dimension $m \times n$. By giving this input to an encoder reduce this input to a k dimensional vector while mostly conserving the information it contains. This k dimensional vector is usually called *encoding* or *embedding*.

After encoding the input to a low dimensional representation, we give it to a decoder. Decoder takes this encoding as input and use it to create a sequence relevant to the input sequence. Encoding acts as a condition to the decoder. Decoder generates the first token of the sequence using the encoding. After that decoder creates its own internal state to generate the second token and so on. This process continues until we reach the maximum possible output sequence length (which is determined by us) or the decoder generates the special token <eos> which stands for '**e**nd **o**f **s**equence'.

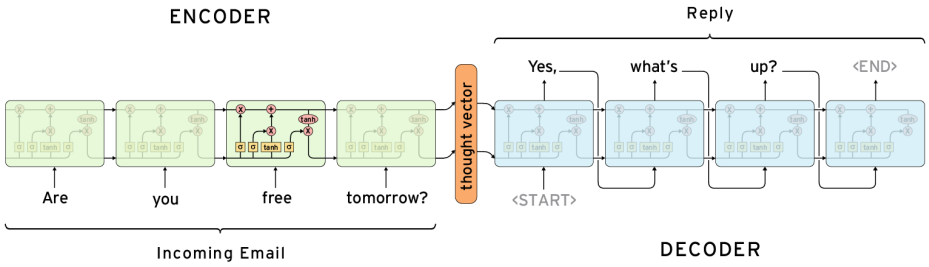

Figure 2.8: Encoder Decoder
[10]

2.11 Attention

We've seen in the previous section, encoder takes the input text and encodes it into vector representation of fixed length k. This may not always fully represent the input text because length and density of the text can vary. Therefore, it can be logical to create a number of representations that changes according to the length of the input. This way, decoder would better understand the input and generate meaningful outputs. Attention mechanism is developed to do exactly this. There are couple of proposed attention mechanisms but their main idea is same: encode the input into a vector sequence and give their weighted sum to the decoder. When decoder generates a sequence, through attention mechanism, it receives some scores of the input elements showing the relevance of that element to a specific output. Therefore, decoder knows how much "attention" it must give to every input element. Attention score is also called *alignment score*. These scores are calculated between every jth element of the input and ith element of output. If the input length is m and the output length is n, then $m \times n$ scores are calculated.

2.11.1 Bahdanau Attention

One popular attention mechanism is Bahdanau atttention. In Bahdanau attention, alignment scores are calculated between the decoder's hidden state and the encoder's output [11]. Score function that Bahdanau attention uses is as follows:

$$a(s_{i-1}, h_j) = v^T \tanh(W_a s_{i-1} + U_a h_j) \tag{2.38}$$

where s_{i-1} is the hidden state of the decoder and h_j is the output of the encoder. v, W_a and U_a are the parameters of the attention mechanism that is to be optimized while training.

These scores are calculated for every i and j pair and normalized as:

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T_x} exp(e_{ik})} \tag{2.39}$$

where T_x is the maximum length of the input of the encoder. Lastly, by using the normalized scores and the output of the encoder, we obtain a *context vector*:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{2.40}$$

This context vector is given as input to the decoder, along with the generated question words. Therefore, decoder is better conditioned to generate a relevant question according to the input text.

3 Natural Language Processing

Natural language processing (NLP) is the study of interactions of computers and human languages. Specifically, its research is analyzing and understanding the human language using computers [12].

3.1 NLP Tasks

Popular NLP tasks are morphological analysis, syntactic analysis, speech recognition, text processing. In morphological analysis, words are processed into *morphemes* to understand sentences. Morpheme is the smallest unit in a language that is meaningful to humans [13]. For example, the word "going" can be processed as "go" + "ing". Syntactic analysis is the research of analyzing sentences using the grammar rules of a language. Speech recognition is the task of converting digital voice to meaningful text.

3.2 NLP and Deep Learning

In recent years, NLP and deep learning have started to be used often. Various neural network architectures for NLP tasks are developed. State-of-the-art models for NLP tasks are now usually the deep learning models. Architectures such as recurrent neural networks, *transformers* are the most popular ones that are used in deep learning projects.

3.3 Word Representations

In deep learning tasks related to nlp, we first model the language we are working on. Then, we attempt to solve the problem that's related to our task. To model a language, we need to learn the relationship between words. If we represent every word as a vector in an n-dimensional space in a way that the words that are related to each other are closer in distance, we can use these vectors as features to our model so that it can solve our task by knowing the similarities between the words. This idea of representing a word as a vector is called *word embedding*. While there are frequency based methods for obtaining word vectors, model based word embeddings that are trained on a large corpus give much higher performance on our tasks. There are couple word embedding models which are popularly used: Word2Vec and GloVe.

3.3.1 Word2Vec

Word2Vec helps us learn word embeddings by using a simple model that can be trained on a corpus. Word2Vec is actually consists of 2 proposed models to learn word representations: Continuous Bag of Words (CBOW) and Continuous Skip-gram. CBOW works by predicting a word by looking at its surrounding words. Number of words to look at defines the window size. Skip-gram model works opposite. It tries to predict the surrounding words of a single word. Both models are simple feed forward neural networks with a single hidden layer. But, if trained on a large corpus, they can be very effective on capturing syntactic and semantic relationships between words. For example, $vector("king") - vector("man") + vector("woman")$ gives a vector such that its closest vector is $vector("queen")$ [14].

3.3.2 GloVe

GloVe, proposed by Pennington, Jeffrey et. al., leverages the advantages of matrix factorization and local context methods [15]. They suggest that, while factorization based methods, such as LSA (Latent Semantic Analysis) uses the statistical properties of the corpus, they don't do really well on learning relationships between words. Also, while local context methods such as Skip-gram model in Word2Vec, capture the analogies between words, they don't take advantage of global statistics of the corpus.

In GloVe, a *global co-occurence matrix*, $\mathbf{X}$ is built and used. $\mathbf{X}_{ij}$ denotes how many times the word j occurs in the context i. Also, co-occurence probability $P_{ij} = P(j|i)$ are calculated. $P_i j$ is calculated as follows:

$$P_{ij} = \frac{X_{ij}}{\sum_k X_{ik}} \tag{3.1}$$

Using these, they propose a regression model:

$$\mathbf{w}_i^T \tilde{\mathbf{w}}_k + b_i + \tilde{b}_k = \log \mathbf{X}_{ik} \tag{3.2}$$

where $\mathbf{w}$ and $\tilde{\mathbf{w}}$ are word vectors that the model tries to learn. b and $\tilde{b}$ are bias terms.

Objective of this model is a least squares regression problem and is the following:

$$J = \sum_{i,j=1}^{V} f(\mathbf{X}_{ij})(\mathbf{w}_i^T \tilde{\mathbf{w}}_j + b_i + \tilde{b}_j - \log \mathbf{X}_{ij})^2 \tag{3.3}$$

where $f(x)$ is weighting function that penalizes the errors of the similarity between words that are most co-occured.

4 Project

In the project, our main was to provide an educative mobile application that its content is generated using deep learning models.

In this section, question generation part of the project is explained in the light of the theoretical background of the algorithms and models presented in the previous sections.

4.1 Question Generation

Question generation for reading compherension can be defined as generating questions from a given informative text that test the readers' knowledge about the information given in the input text. This task can be easily achieved by us humans and it can be a repetitive and time consuming task. With deep learning and advanced nlp techniques, we can build a model that automatically generates questions from a given text. In the following sections, we present our attempt to achieve this task.

4.2 Related Work

So far, several studies have been done in recent years that used deep learning. Xinya Du et. al. [16] have been used encoder-decoder LSTM architecture along with attention mechanism. Also Qingyu Zhou et. al. have been used encoder-decoder LSTM architecture with pointing mechanism to handle rare words from source text [17].

Also, there are more sophisticated development such as Preksha Nema et. al. that uses reinforcement learning to tackle this task. [18]

4.3 Data Set

Finding a proper dataset is crucial for every machine learning task. After some research, SQuAD (Stanford Question Answering Data Set) is found [19]. SQuAD contains more than 100,000 questions and answers along with their source texts. This question and answer pairs are generated from more than 500 articles. SQUAD is proved to be a useful benchmark for this task.

Data set consists of three parts: training, development and test sets. Training and development sets are public and open to everyone but there is no access to the test set.

Therefore, only training and development sets are used. In the data set, there are text passages and question related to the text. With questions, we have the information that the question can be answered by reading the corresponding text. We also have the answers of the questions if they exist. With the answers, we have answer position of the question in the text.

In the project, training and development sets are merged. Then, this merged data set are splitted such that we have %95 training set, %5 development set, %5 test set. This is done for increasing the number of examples in our training set. Also, before splitting, the observations that its question can not be answered are removed from the data set.

4.4 Data Analysis and Preprocessing

After splitting the data set, we have the following number of observations for training, development and test sets, respectively: 82479, 4233, 4233.

Preprocessing steps are done for each of these sets.

First, we have observed that the input texts are too long to give to a model. Using the answer position feature in SQuAD, sentence that contains the answer is detected for each question. We decided to use only the sentence that contains the answer from the input text. This has reduced the problem of large input texts. After that, all the letters in the data set are converted to lower case. Symbols that represent specific meanings (degree symbol and currency symbols etc.) are converted to their word equivalent. Contractions such as *haven't* or *don't* are expanded. Non-alphanumeric characters are removed. After these steps, texts are tokenized into "words". All these preprocessing steps are applied to input texts, question texts and answer texts.

Lastly, special tokens are added to beginning and end of the question texts. These are `<sos>` and `eos` tokens. These tokens will help the model to know where the output starts and where it ends.

Training time of our model depends on the length of our inputs. Therefore, working with shorter texts will reduce the training time of the model and helps us do more experiments in shorter time. Length of the extracted input sentences are analyzed. In Figure 4.1, distribution of the input sentences' lengths are visualized. It can be seen that input sentences generally consists of 50 or less words. Also, in Figure 4.2, we see that the questions texts are usually below or equal to 20 words. So, we removed the observations that has input text greater than 50 words or has question text greater than 20 words. After this removal, we had 73,000+ observations in our training set and around 3800 observations in our development and test sets.

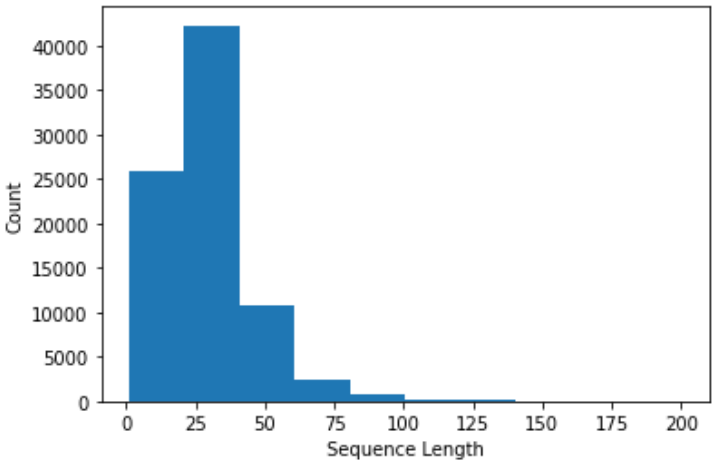

Figure 4.1: Distribution of lengths of the input texts

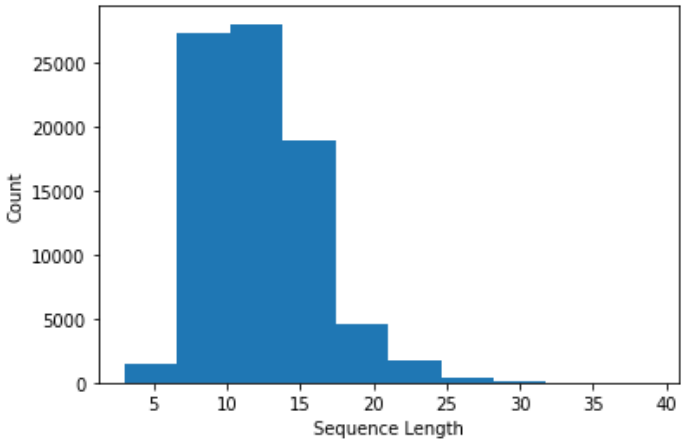

Figure 4.2: Distribution of lengths of the question texts

Having done the preprocessing steps and tokenized the texts into words, vocabulary of the training set is built. We have seen that training set has around 60,000+ unique words in the input texts and 33,000 unique words in the question texts. We chose the 53,000 most common words in the input texts and replaced other words with `<unk>` token which stands for *unknown*. As the input texts, we applied the same procedure to the question texts. We found the 28,000 most common words and replaced other words with <unk>. Doing this will prevent model from being too dependent on rare words. Also, it will make the model to generate <unk> token when it encounters a word that it has not seen rather than generating an irrevelant word.

As the final preprocessing steps, all words are assigned to an index and input texts and question texts are padded. Padding is applied so that all input texts have the length of 50 and all question texts have the length of 20. Padding works by adding 0's to the

beginning or the end of a sequence so that all sequneces have the same length. In this case, we applied padding to the end of the sequences.

4.5 Pretrained Word Embeddings

In the project, pretrained word embeddings are used in the first layer of the model. Pretrained 300 dimensional, 1.9M GloVe word embeddings are used. These vectors are learned on a large data set which contains webpages' contents which is called *Common Crawl*.

4.6 Model

Two different models are trained throughout the project. These two models only differ in the inputs. Their architecture is same. An encoder-decoder architecture is used in the project. First, only the input sentence is given to the encoder, and the question sentence is given to the decoder. In the second model, encoder received both the input text and answer of the question. That way, decoder was better targeted to generate the correct question.

Encoder has an embedding layer which its weights are initialized using the GloVe embeddings. Words that are not present in the GloVe are randomly initialized. Encoder has a biredictional LSTM layer with 1024 units after the embedding layer. Dropout with rate %20 applied to the LSTM outputs. Also, *recurrent dropout* of %10 is applied between the LSTM timesteps. Recurrent dropout randomly zeros out the connections between the timesteps.

Decoder also has an embedding layer and a biredictional LSTM layer with 1024 hidden units. Embedding layer has been initialized with GloVe vectors. In addition to these, decoder has an attention layer which uses Bahdanau attention mechanism. In the attention layers, L_2 regularization with $\lambda = 10^{-4}$ is applied to the weights. Also, dropout with %10 rate is applied on the attention outputs. Attention outputs and the previously generated token are given to the LSTM. Output of the LSTM is then given to a final fully connected layer to predict the next word of the question to be generated.

4.6.1 Training

As we discussed in the previous sections, two different models are trained. First model, received only the input text and question. Second model received input text, answer

and question. Both models are trained using the Adam optimizer with learning rate $1e-3$, $\beta_1 = 0.9$, $\beta_2 = 0.999$. We clip the norms of the gradients that are greater than 1. We used batch size of 64. We stopped the training proces when validation loss increased for 2 consecutive epochs. Model without answers are trained for 9 epochs and model with answers are trained for 10 epochs.

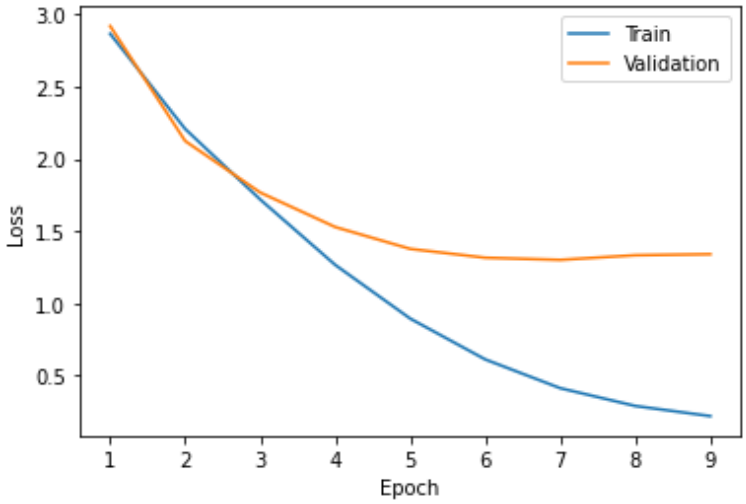

Figure 4.3: Validation loss across epochs of the model without answers

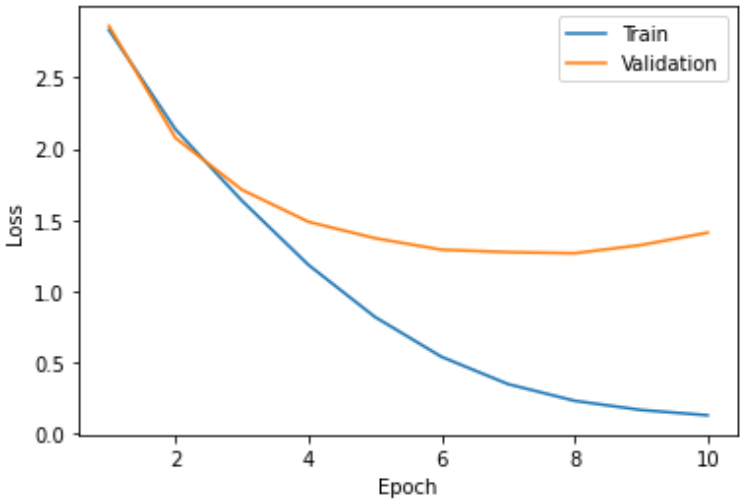

Figure 4.4: Validation loss across epochs of the model with answers

Models are trained in a way that for each example, a question is generated word by word. Then the loss across each timestep is averaged. There are 20 timesteps because the maximum question length in our data set is 20. For training, we used a method called *teacher forcing*. Teacher forcing works by, in training time, in each timestep we give the "true" word to the decoder rather than giving the previously generated word. This make the training process faster to converge because at the beginning of

the training, model is very likely to generate a incorrect or irrevelant output, so giving the wrong output to the next timestep would make the training process harder.

4.7 Inference

In inference, namely the question generation part, 2 different method is applied. Model is initialized with the input text and the <sos> token for the question. After that, model generates probabilities for each of the word in the dictionary. In the first method, word with the highest probability is selected and that word is fed into the decoder to generate the next word and so on. This process is repeated until the model generates the <eos> or it generates total of 20 words. This method is called *greedy search* and it may not generates the best result because it looks at only one candidate output by taking the most likely word at each timestep and it may miss other possible candidates.

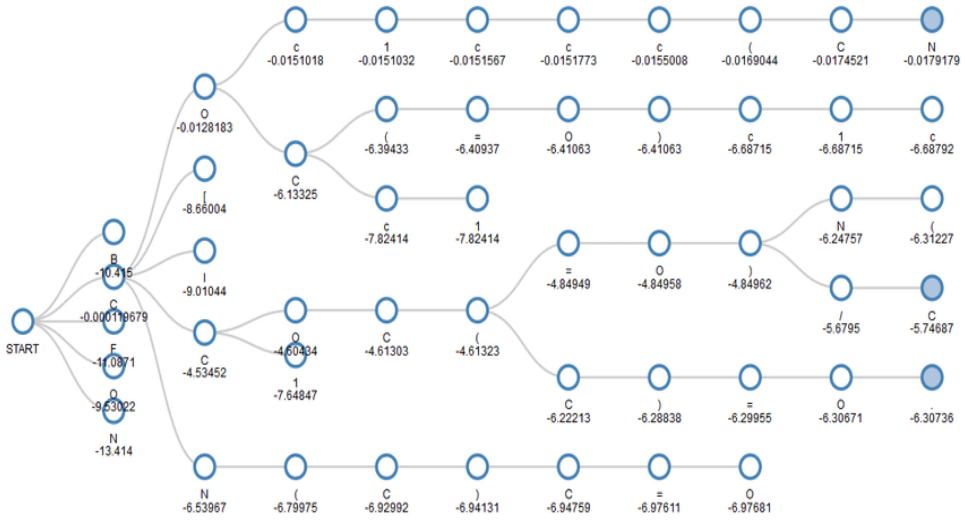

Figure 4.5: Beam Search Example (with beam width = 5)
[20]

Beam search is used as an alternative to the greedy search. Beam search evaluates multiple candidates at each timestep. Number of candidates to keep at each iteration is determined by an hyperparameter called *beam width*. In this algorithm, at time t, rather than taking the word with highest probability, we take the most likely n (beam width) candidate sentences with highest score and move on with the next timestep. In the next timestep, we again choose n candidates. This process is stopped when all candidates reached to the <eos> token or have 20 words. Score is a joint probability calculated by multiplying probabilities of the generated words which can be expressed

as:

$$P(\mathbf{y},\mathbf{x}) = P(y_0|\mathbf{x})P(y_1|\mathbf{x},y_0)P(y_2|\mathbf{x},y_0,y_1)\dots P(y_{T_y}|\mathbf{x},y_0,\dots,y_{T_y-1}) \tag{4.1}$$

where T_y is the maximum output length and $y = [y_0, y_1, \dots, y_{T_y}]^T$ is the output sentence.

So, greedy search works as taking the most likely word at each timestep. It finds a suboptimal solution because it does not take into account the overall probability but only considers the conditional word probability. On the other hand, beam search tries to find the best candidate sentence that has the maximum $P(\mathbf{y},\mathbf{x})$.

4.8 Evaluation

We evaluated the models we trained on the test set. We used beam search with beam width of 3 to generate the questions. Two different metrics are reported: BLEU and ROUGE.

4.8.1 BLEU

BLEU score compares the n-grams in the reference text and the generated text. A higher BLEU score is obtained if the ratio of the matching n-grams between the generated text and the reference text is high. Precision is calculated for every type of n-gram where n = 1, 2, ..., N and the calculated precisions are averaged with predetermined weights. Resulting value is multiplied by a *brevity penalty* term. Brevity penalty is applied to penalize the candidate (generated) texts that are longer than their reference texts [21]. Brevity penalty term is given as:

$$BP = \begin{cases} (y^{(i)} - \hat{y}^{(i)})^2 & \text{if c > r} \\ e^{1-r/c} & \text{otherwise.} \end{cases} \tag{4.2}$$

where r is the length of the reference text and c is the length of the generated text.

$$BLEU = BP \,.\, exp(\sum_{i=1}^{N} w_n p_n) \tag{4.3}$$

In the evalution we calculated precisions up to 4-grams and gave them equal weights, which is $\frac{1}{4}$.

4.8.2 ROUGE

ROUGE is another common set of metrics that are used to evaulate machine translation, text summarization or any other related models. ROUGE consists of several metrics that measures the success of a generated text related to its reference. ROUGE-N, counts the matching N-grams in the text. ROUGE-L metric considers sequential structure of the text by finding the longest common subsequences between the sentences.

4.8.3 Results

Calculated metrics of the two models are reported in the table below. Note that the F1-score of the ROUGE metrics are given.

Model	BLEU	ROUGE-1	ROUGE-2	ROUGE-L
without answers	46.8	46.1	30.8	45.0
with answers	51.7	53.0	38.2	52.1

Table 4.1: Metrics on the test set

4.9 Software Tools

All the project is developed in *Python* programming language.

Tensorflow and *Keras* frameworks are used to build and train the models. Also, we leveraged from Keras in the preprocessing steps for indexing the words and padding the inputs. Built-in *re* (regular expressions) package and *nltk* library are used for text preprocessing. *pandas* library is used for storing our data in a structured way.

4.10 Other Parts of the Project

As stated before, this project is part of an educative mobile application project. Besides question generation, there are text summarization and app development parts in the project. In this section, we give brief descriptions and results of these parts.

4.10.1 Text Summarization

Ayşe Duman was responsible for text summarization. In this part, our aim was to summarize wikipedia articles for displaying on the application. Wikipedia articles are obtained from an API provided by wikipedia. First, RNNs are researched and developed to do abstractive text summarization but the input texts was too long and we hadn't enough computation power. After this attempt, extractive and unsupervised text summarization techniques are developed. Document and sentence embedding techniques are researched. Doc2Vec and BERT models are applied to create embeddings of the sentences in the texts. Then, these embeddings are clustered using K-Means clustering algorithm. After that, the closest sentences to the cluster centers are selected and are merged to create the summary.

4.10.2 Mobile Application Development

Eda Nur Ersu was responsible for developing the graphical interface, logic of the application and the deployment. Application consists of several pages. First there is login screen. User can login via his/her google account or not choose to login and continue to the application. After this step, there is the main screen which displays the categories of the subjects present in the application. User can also search for a specific subject by typing it on a search bar. After typing, user then can opens an article (generated by the summarization model) page to read about a specific subject. Having read the article, user can take the quiz generated by our question generation model. Also, there is a profile page in which the quiz scores are displayed. Lastly, there is a page which shows the saved articles of the user. Application is named as *RoboSummy* [1], expressing its main idea, summaries generated by a "robot".

[1] It can be downloaded from google play .

5 Conclusion and Future Work

In this study, we discovered many concepts such as machine learning, deep learning, sequence modelling, natural language processing etc. Specifically, we reviewed the methods for working with text data to generate novel text. We have seen encoder-decoder architecture and how it can be applied to question generation task. After that, we gave the details of the models we built and described the experimental process and results of our study. Lastly, we talked about the overall project and briefly described the main parts of the project.

There are lots of things to do for future work. RNN model presented in this study can certainly be improved by trying other hyperparameters and architectures. Also, *transformer* based models such as *BERT* (Bidirectional Encoder Representations from Transformers) can be tried to further improve the performance. In addition to that, other data sets for question generation can be used. Also, data can be collected for other languages such as Turkish and another model can be trained on it.

References

[1] Ethem Alpaydın. *Introduction to Machine Learning*. The MIT Press, 2004.

[2] Wikipedia. Sigmoid function. https://en.wikipedia.org/wiki/Sigmoid_function.

[3] Lin Yu-Hsiu et. al. Chen Yung-Yao. Design and implementation of cloud analytics-assisted smart power meters considering advanced artificial intelligence as edge analytics in demand-side management for smart homes. *Sensors*, (19):3, 2019.

[4] Wikipedia. Artificial neural network. https://en.wikipedia.org/wiki/Artificial_neural_network.

[5] Wikipedia. Recurrent neural network. https://en.wikipedia.org/wiki/Recurrent_neural_network.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[8] Colah's Blog. Understanding lstm networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs.

[9] Wikipedia. Gated recurrent unit. https://en.wikipedia.org/wiki/Gated_recurrent_unit.

[10] Medium. Sequence to sequence tutorial. https://towardsdatascience.com/sequence-to-sequence-tutorial-4fde3ee798d8.

[11] Dzmitry Bahdanau, Kyunghyun Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *ArXiv*, 1409, 09 2014.

[12] Wikipedia. Natural language processing. https://en.wikipedia.org/wiki/Natural_language_processing.

[13] Wikipedia. Morpheme. https://en.wikipedia.org/wiki/Morpheme.

[14] Tomas Mikolov, G.s Corrado, Kai Chen, and Jeffrey Dean. Efficient estimation of word representations in vector space. pages 1–12, 01 2013.

[15] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. volume 14, pages 1532–1543, 01 2014.

[16] Junru et. al. Du, Xinya Shao. Learning to ask: Neural question generation for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1342–1352, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[17] Nan et. al. Zhou, Qingyu Yang. Neural question generation from text: A preliminary study. In *Natural Language Processing and Chinese Computing*, pages 662–671. Springer International Publishing, 2018.

[18] Akash Kumar et. al. Nema, Preksha Mohankumar. Let's ask again: Refine network for automatic question generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3314–3323. Association for Computational Linguistics, 2019.

[19] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. pages 784–789, 01 2018.

[20] Bowen Liu, Bharath Ramsundar, Prasad Kawthekar, Jade Shi, Joseph Gomes, Quang Nguyen, Stephen Ho, Jack Sloane, Paul Wender, and Vijay Pande. Retrosynthetic reaction prediction using neural sequence-to-sequence models. *ACS Central Science*, 3, 06 2017.

[21] Kishore Papineni, Salim Roukos, Todd Ward, and Wei Jing Zhu. Bleu: a method for automatic evaluation of machine translation. 10 2002.

Resume

Full Name	Tuğrul Hasan Karabulut
Date of Birth	05.12.1998
Place of Birth	Istanbul
Internships	Jr. Software Developer at bidolubaski.com (1 year and 8 months)