MACQUARIE University

ITIC systems

Automatic Exam Marking Project

An Introduction to

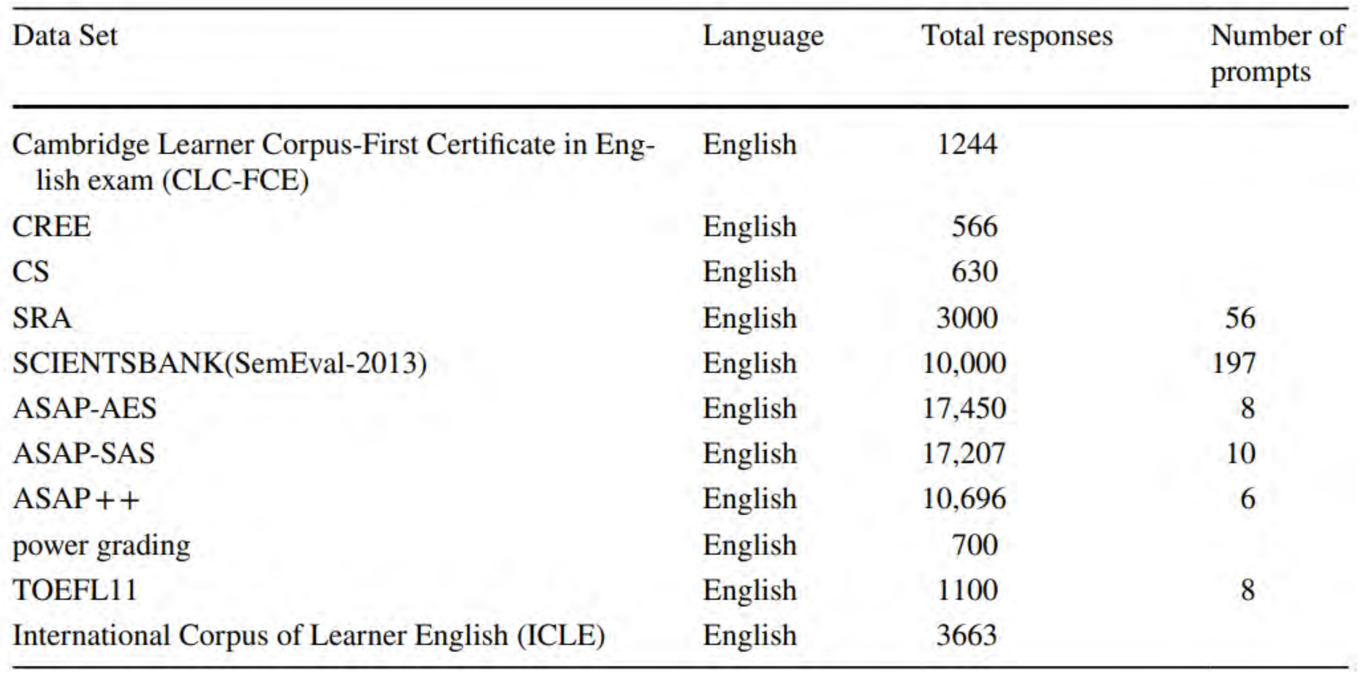
Using GAN for Essay Dataset Generation
&
ML Methods for Essay Scoring

Research questions

datasets	Features	Evaluation Metrics	ML Techniques	challenges
what are the datasets available for research on automated essay grading?	what are the features extracted for the assessment of essays?	which are the evaluation metrics for measuring the accuracy of algorithms?	What are the ML techniques used for AES, and how implemented?	What are the challenges and limitations in the current research?
Question1	Question2	Question3	Question4	Question5

Q1-Database

The answer to the question can provide a list of the available datasets, their domain, and access to the datasets. It also provides a number of essays and corresponding prompts.



Data Set	Language	Total responses	Number of prompts
Cambridge Learner Corpus-First Certificate in English exam (CLC-FCE)	English	1244	
CREE	English	566	
CS	English	630	
SRA	English	3000	56
SCIENTSBANK(SemEval-2013)	English	10,000	197
ASAP-AES	English	17,450	8
ASAP-SAS	English	17,207	10
ASAP++	English	10,696	6
power grading	English	700	
TOEFL11	English	1100	8
International Corpus of Learner English (ICLE)	English	3663	

Research questions

datasets

what are the datasets available for research on automated essay grading?

Question1

Features

what are the features extracted for the assessment of essays?

Question2

Evaluation Metrics

which are the evaluation metrics for measuring the accuracy of algorithms?

Question3

ML Techniques

What are the ML techniques used for AES, and how implemented?

Question4

challenges

What are the challenges and limitations in the current research?

Question5

ITIC systems

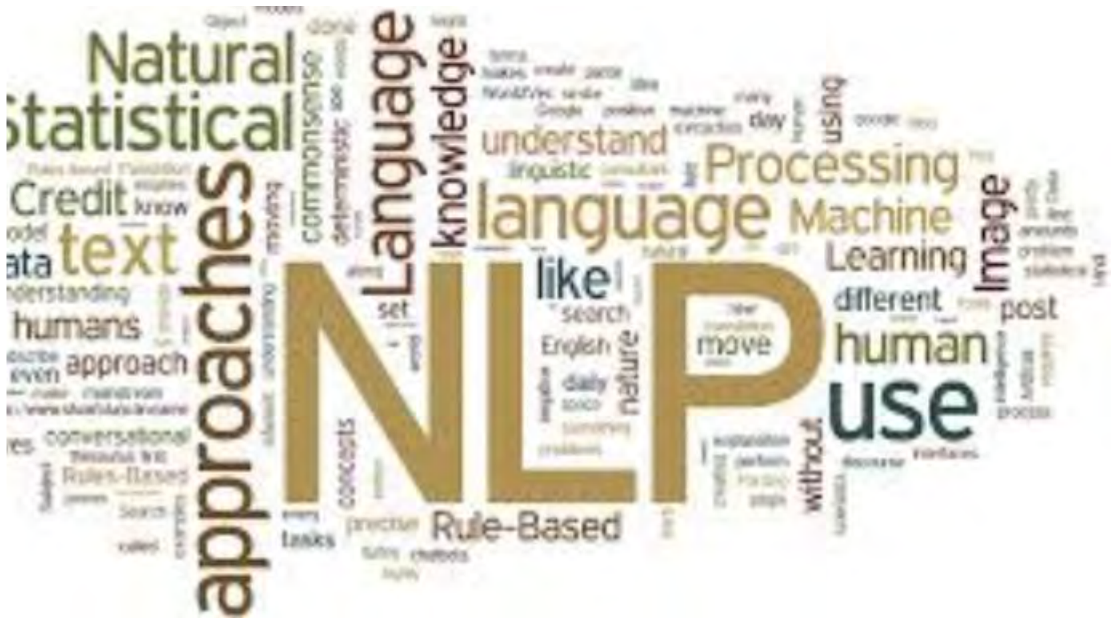
Q2-Features

Features play a major role in the neural network and other supervised Machine Learning approaches. The automatic essay grading systems scores student essays based on different types of features, which play a prominent role in training the models. Based on their syntax and semantics and they are categorized into three groups:

Statistical based features	Style-based (Syntax) features	Content-based features
Essay length with respect to the number of words	Sentence structure	Cohesion between sentences
Essay length with respect to sentence	POS	Overlapping (prompt)
Average sentence length	Punctuation	Relevance of information
Average word length	Grammatical	Semantic role of words
N-gram	Logical operators	Correctness
	Vocabulary	Consistency
		Sentence expressing key concepts

Vectorization: First Step of each Technique

- Vectorization is jargon for a classic approach of converting input data from its raw format (i.e. text) into vectors of real numbers which is the format that ML models support. This approach has been there ever since computers were first built, it has worked wonderfully across various domains, and it's now used in NLP.
- In Machine Learning, vectorization is a step in feature extraction. The idea is to get some distinct features out of the text for the model to train on, by converting text to numerical vectors.
- There are plenty of ways to perform vectorization, as we'll see shortly, ranging from naive binary term occurrence features to advanced context-aware feature representations. Depending on the use-case and the model, any one of them might be able to do the required task



Vectorization techniques

How to transform a text

01 Bag of Words

02 TF-IDF

03 Word2Vec

04 Glove

05 Fast Text

Bag of Words

- Most simple of all the techniques out there. It involves three operations:
- **Tokenization**

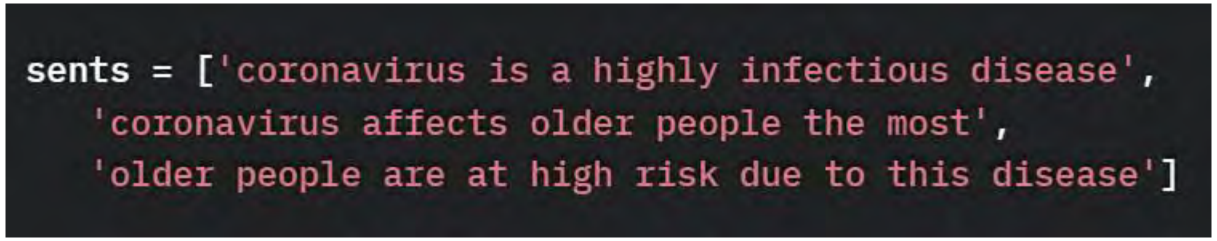
First, the input text is tokenized. A sentence is represented as a list of its constituent words, and it's done for all the input sentences.

- **Vocabulary creation**

Of all the obtained tokenized words, only unique words are selected to create the vocabulary and then sorted by alphabetical order.

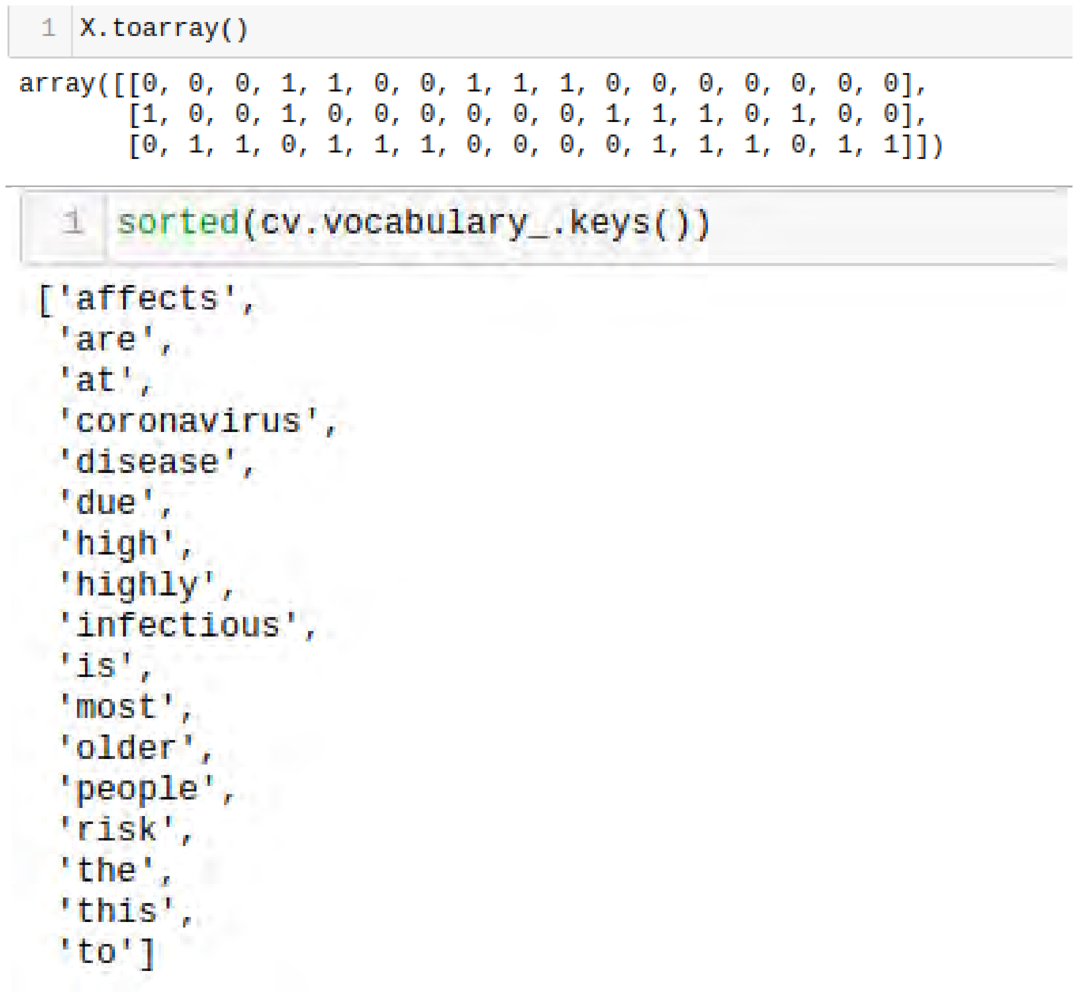
- **Vector creation**

Finally, a sparse matrix is created for the input, out of the frequency of vocabulary words. In this sparse matrix, each row is a sentence vector whose length (the columns of the matrix) is equal to the size of the vocabulary.



```
sents = ['coronavirus is a highly infectious disease',
   'coronavirus affects older people the most',
   'older people are at high risk due to this disease']
```

- You can see that every row is the associated vector representation of respective sentences.
- The length of each vector is equal to the length of vocabulary.
- Every member of the list represents the frequency of the associated word as present in sorted vocabulary.



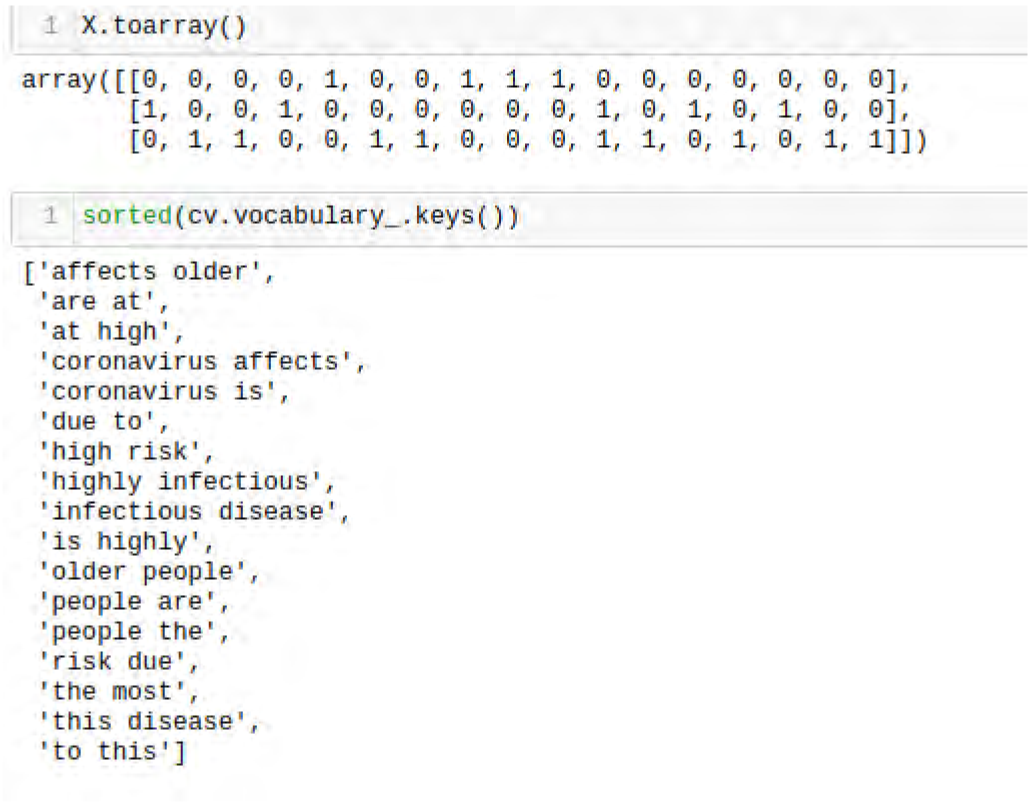
```
X.toarray()
```

```
array([[0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0],
       [0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1]])
```

```
sorted(cv.vocabulary_.keys())
```

```
['affects',
 'are',
 'at',
 'coronavirus',
 'disease',
 'due',
 'high',
 'highly',
 'infectious',
 'is',
 'most',
 'older',
 'people',
 'risk',
 'the',
 'this',
 'to']
```

- In the above example, we only considered single words as features as visible in the vocabulary keys, i.e. it’s a unigram representation. This can be tweaked to consider n-gram features.
- Let’s say we wanted to consider a bigram representation of our input. It can be achieved by simply changing the default argument while instantiating the CountVectorizer object.
- In that case, our vectors & vocabulary would look like this.
- Thus we can manipulate the features any way we want. In fact, we can also combine unigrams, bigrams, trigrams, and more, to form feature space.
- Although we’ve used sklearn to build a Bag of Words model here, it can be implemented in a number of ways, with libraries like Keras, Gensim, and others. You can also write your own implementation of Bag of Words quite easily.
- This is a simple, yet effective text encoding technique and can get the job done a number of times.



```
X.toarray()
```

```
array([[0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0],
       [0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1]])
```

```
sorted(cv.vocabulary_.keys())
```

```
['affects older',
 'are at',
 'at high',
 'coronavirus affects',
 'coronavirus is',
 'due to',
 'high risk',
 'highly infectious',
 'infectious disease',
 'is highly',
 'older people',
 'people are',
 'people the',
 'risk due',
 'the most',
 'this disease',
 'to this']
```

TF-IDF

- TF-IDF or Term Frequency–Inverse Document Frequency, is a numerical statistic that’s intended to reflect how important a word is to a document. Although it’s another frequency-based method, it’s not as naive as Bag of Words.
- **How does TF-IDF improve over Bag of Words?**
- In Bag of Words, we witnessed how vectorization was just concerned with the frequency of vocabulary words in a given document. As a result, articles, prepositions, and conjunctions which don’t contribute a lot to the meaning get as much importance as, say, adjectives.
- TF-IDF helps us to overcome this issue. Words that get repeated too often don’t overpower less frequent but important words. It has two parts: **TF and IDF**

TF

- TF stands for **Term Frequency**. It can be understood as a normalized frequency score. It is calculated via the following formula:

$$TF = \frac{Frequency\ of\ word\ in\ a\ document}{Total\ number\ of\ words\ in\ that\ document}$$

- So one can imagine that this number will always stay ≤ 1, thus we now judge how frequent a word is in the context of all of the words in a document.

IDF

- IDF stands for Inverse Document Frequency, but before we go into IDF, we must make sense of DF – Document Frequency. It’s given by the following formula:

$$DF = \frac{Documents\ containing\ word\ W}{Total\ number\ of\ documents}$$

- DF tells us about the proportion of documents that contain a certain word. So what’s IDF?
- It’s the reciprocal of the Document Frequency, and the final IDF score comes out of the following formula:

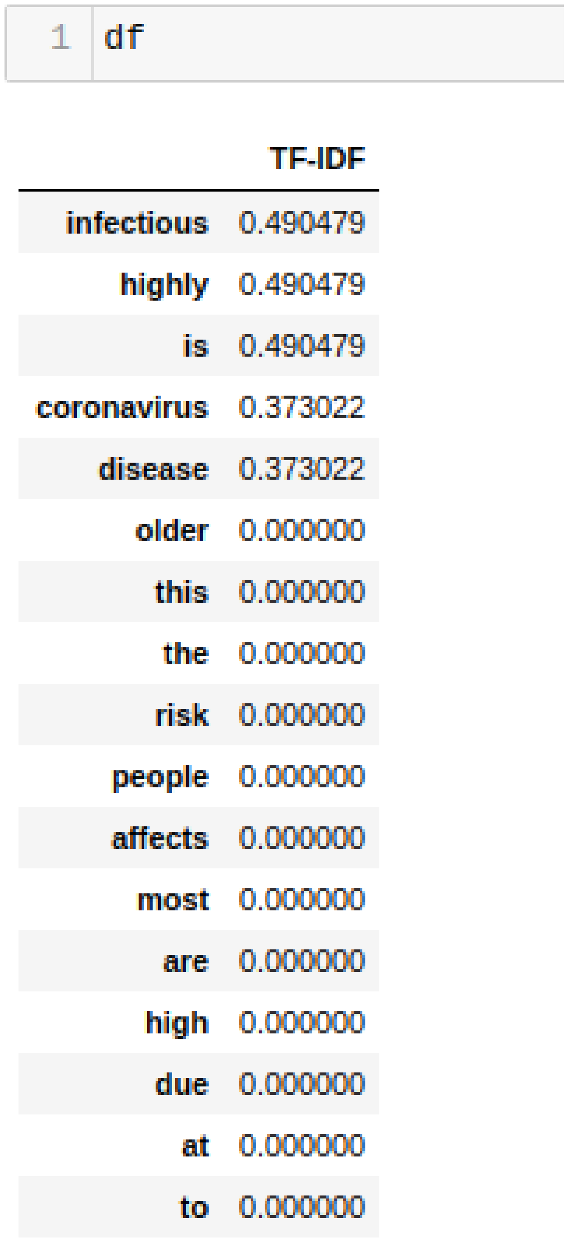
$$IDF = \log(\frac{Total\ number\ of\ documents}{Documents\ containing\ word\ W})$$

IDF

- Why inverse the DF? Just as we discussed above, the intuition behind it is that the more common a word is across all documents, the lesser its importance is for the current document.
- A logarithm is taken to dampen the effect of IDF in the final calculation. The final TF-IDF score comes out to be:

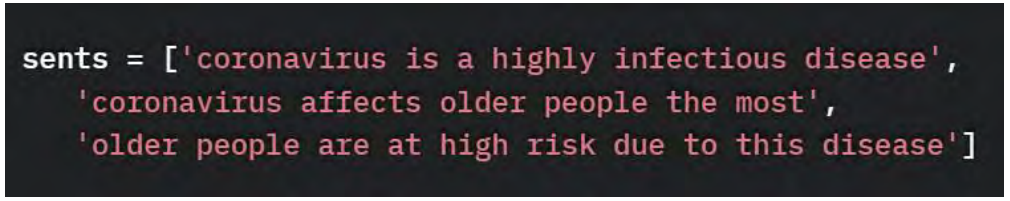
$$TF - IDF = TF * IDF$$

- This is how TF-IDF manages to incorporate the significance of a word. The higher the score, the more important that word is.



```
df
```

```
               TF-IDF
 infectious  0.490479
     highly  0.490479
         is  0.490479
coronavirus  0.373022
    disease  0.373022
      older  0.000000
       this  0.000000
        the  0.000000
       risk  0.000000
     people  0.000000
    affects  0.000000
       most  0.000000
        are  0.000000
       high  0.000000
        due  0.000000
         at  0.000000
         to  0.000000
```



```
sents = ['coronavirus is a highly infectious disease',
    'coronavirus affects older people the most',
    'older people are at high risk due to this disease']
```

- So, according to TF-IDF, the word ‘infectious’ is the most important feature out there, while many words which would have been used for feature building in a naive approach like Bag of Words, simply amount to 0 here. This is what we wanted all along.
- A few pointers about TF-IDF:
- The concept of n-grams is applicable here as well, we can combine words in groups of 2,3,4, and so on to build our final set of features.
- Along with n-grams, there are also a number of parameters such as min_df, max_df, max_features, sublinear_tf, etc. to play around with. Carefully tuning these parameters can do wonders for your model’s capabilities.
- Despite being so simple, TF-IDF is known to be extensively used in tasks like Information Retrieval to judge which response is the best for a query, especially useful in a chatbot or in Keyword Extraction to determine which word is the most relevant in a document, and thus, you’ll often find yourself banking on the intuitive wisdom of the TF-IDF.
- So far, we’ve seen frequency-based methods for encoding text, now it’s time to take a look at more sophisticated methods which changed the world of word embeddings as we know it, and opened new research opportunities in NLP.

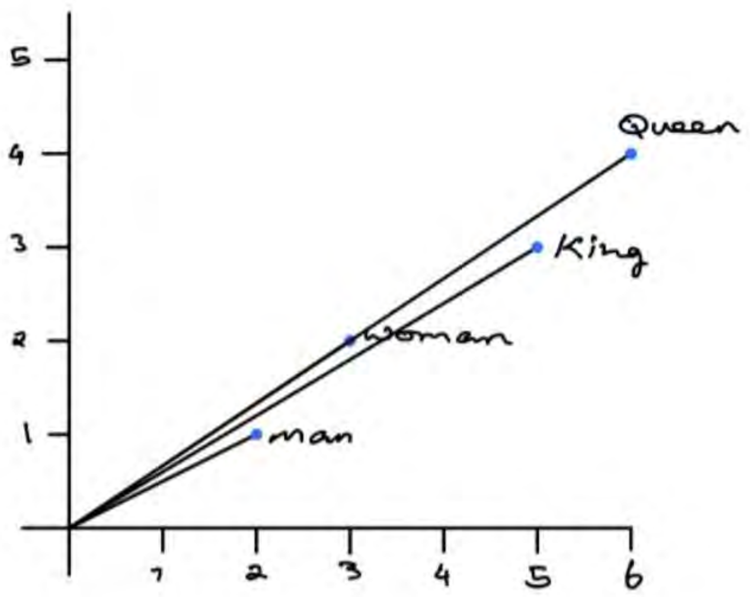
Word2Vec

- This approach was released back in 2013 by Google researchers. In a nutshell, this approach uses the power of a simple Neural Network to generate word embeddings.
- **How does Word2Vec improve over frequency-based methods?**
- In Bag of Words and TF-IDF, we saw how every word was treated as an individual entity, and semantics were completely ignored. With the introduction of Word2Vec, the vector representation of words was said to be contextually aware, probably for the first time ever.
- Perhaps, one of the most famous examples of Word2Vec is the following expression:
- **king – man + woman = queen**
- Since every word is represented as an n-dimensional vector, one can imagine that all of the words are mapped to this n-dimensional space in such a manner that words having similar meanings exist in close proximity to one another in this hyperspace.
- There are mainly two ways to implement Word2Vec, let’s take a look at them one by one:
- **A: Skip-Gram**
- **B. CBOW**

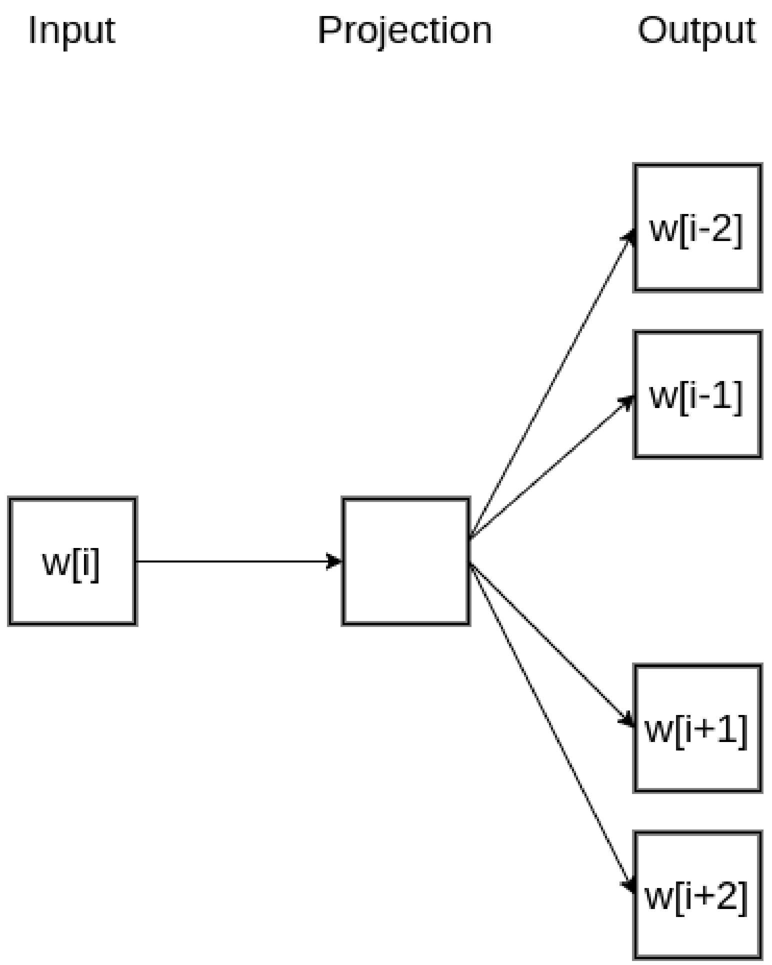
A.Skip-gram

- The skip-gram model is a simple neural network with one hidden layer trained in order to predict the probability of a given word being present when an input word is present. In other word it ask it predicts the context.
- In this architecture, it takes the current word as an input and tries to accurately predict the words before and after this current word. This model essentially tries to learn and predict the context words around the specified input word.
- Based on experiments assessing the accuracy of this model it was found that the prediction quality improves given a large range of word vectors, however it also increases the computational complexity.

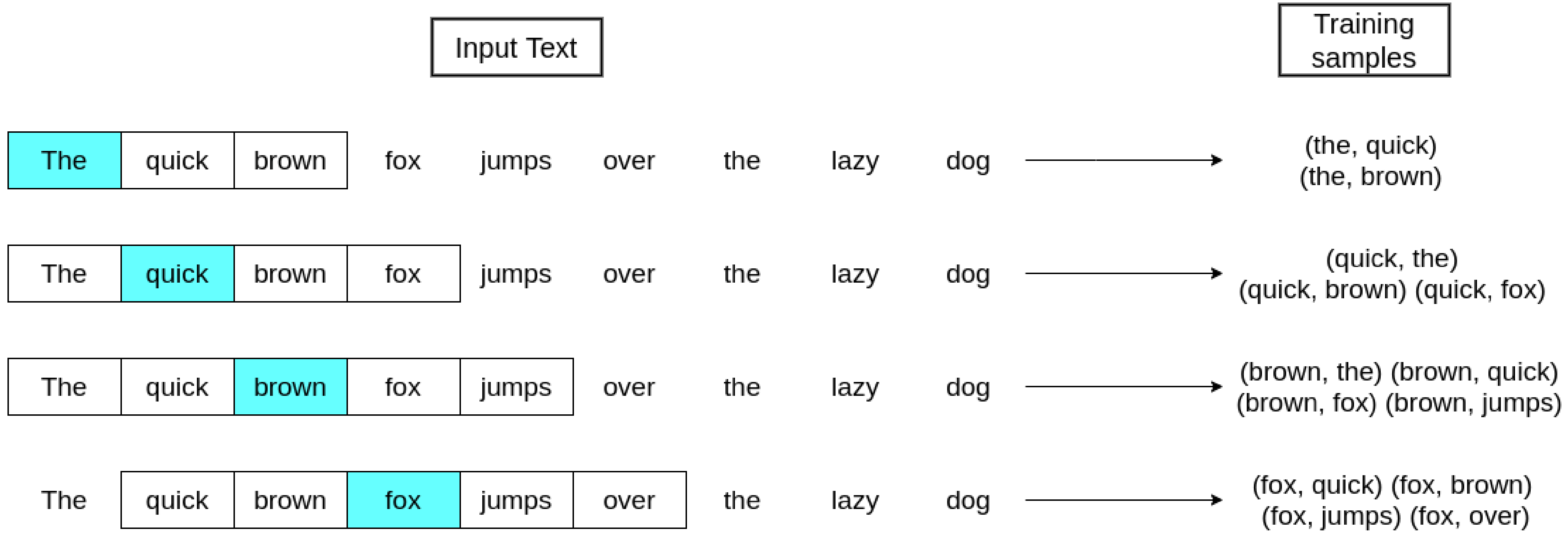
```
King    -    Man    +    Woman    =    Queen
[5,3]   -    [2,1]  +    [3, 2]   =    [6,4]
```



You can see that the words King and Queen are close to each other in position. (Image provided by the author)



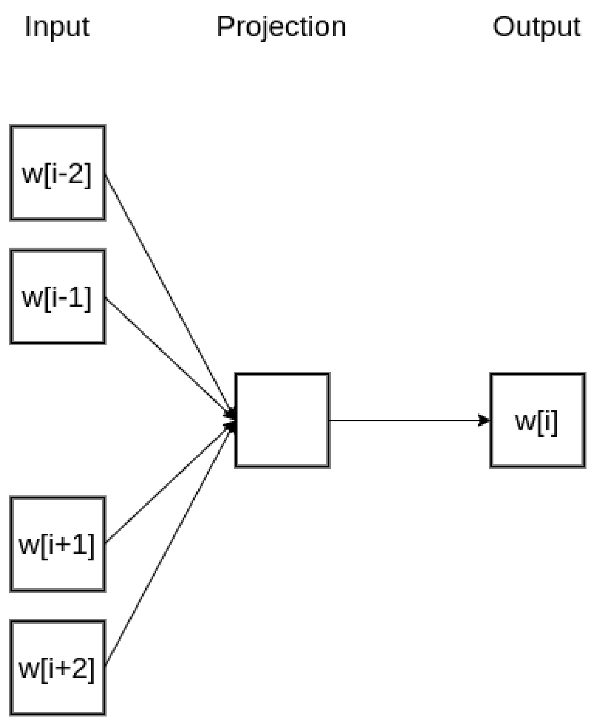
- Here w[i] is the input word at an ‘i’ location in the sentence, and the output contains two preceding words and two succeeding words with respect to ‘i’.Technically, it predicts the probabilities of a word being a context word for the given target word. The output probabilities coming out of the network will tell us how likely it is to find each vocabulary word near our input word.
- This shallow network comprises an input layer, a single hidden layer, and an output layer, we’ll take a look at that shortly. However, the interesting part is, we don’t actually use this trained Neural Network. Instead, the goal is just to learn the weights of the hidden layer while predicting the surrounding words correctly. These weights are the word embeddings.
- How many neighboring words the network is going to predict is determined by a parameter called “window size”. This window extends in both the directions of the word, i.e. to its left and right.
- Let’s say we want to train a skip-gram word2vec model over an input sentence: **“The quick brown fox jumps over the lazy dog”**
- The following image illustrates the training samples that would generate from this sentence with a window size = 2.


Input Text
Training samples
The quick brown fox jumps over the lazy dog
(the, quick)
(the, brown)
The quick brown fox jumps over the lazy dog
(quick, the)
(quick, brown) (quick, fox)
The quick brown fox jumps over the lazy dog
(brown, the) (brown, quick)
(brown, fox) (brown, jumps)
The quick brown fox jumps over the lazy dog
(fox, quick) (fox, brown)
(fox, jumps) (fox, over)

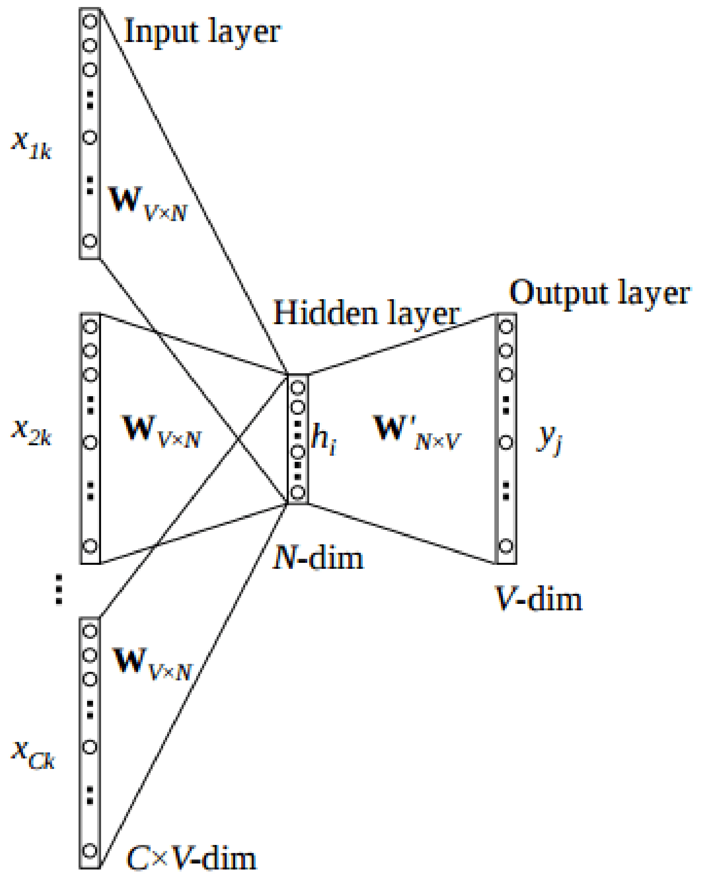
- ‘The’ becomes the first target word and since it’s the first word of the sentence, there are no words to its left, so the window of size 2 only extends to its right resulting in the listed training samples.
- As our target shifts to the next word, the window expands by 1 on left because of the presence of a word on the left of the target.
- Finally, when the target word is somewhere in the middle, training samples get generated as intended.

B. CBOW

- CBOW stands for Continuous Bag of Words. In the CBOW approach instead of predicting the context words, we input them into the model and ask the network to predict the current word. The general idea is shown here:



- You can see that CBOW is the mirror image of the skip-gram approach. All the notations here mean exactly the same thing as they did in the skip-gram, just the approach has been reversed.
- Now, since we already took a deep dive into what skip-gram is and how it works, we won't be repeating the parts that are common in both approaches. Instead, we'll just talk about how CBOW is different from skip-gram in its working. For that, we'll take a rough look at the CBOW model architecture.
- Here's what it looks like:



- The dimension of our hidden layer and output layer stays the same as the skip-gram model. However, just as we read that a CBOW model takes context words as input, here the input is C context words in the form of a one-hot encoded vector of size 1xV each, where V = size of vocabulary, making the entire input CxV dimensional.
- Now, each of these C vectors will be multiplied with the Weights of our hidden layer which are of the shape VxN, where V = vocab size and N = Number of neurons in the hidden layer.
- If you can imagine, this will result in C, 1xN vectors, and all of these C vectors will be averaged element-wise to obtain our final activation for the hidden layer, which then will be fed into our output softmax layer.
- The learned weight between the hidden and output layer makes up the word embedding representation.
- Now if this was a little too overwhelming for you, TLDR for the CBOW model is:
- Because of having multiple context words, averaging is done to calculate hidden layer values. After this, it gets similar to our skip-gram model, and learned word embedding comes from the output layer weights instead of hidden layer weights.

Now let’s talk about the **neural network** which is going to be trained on the aforementioned training samples.

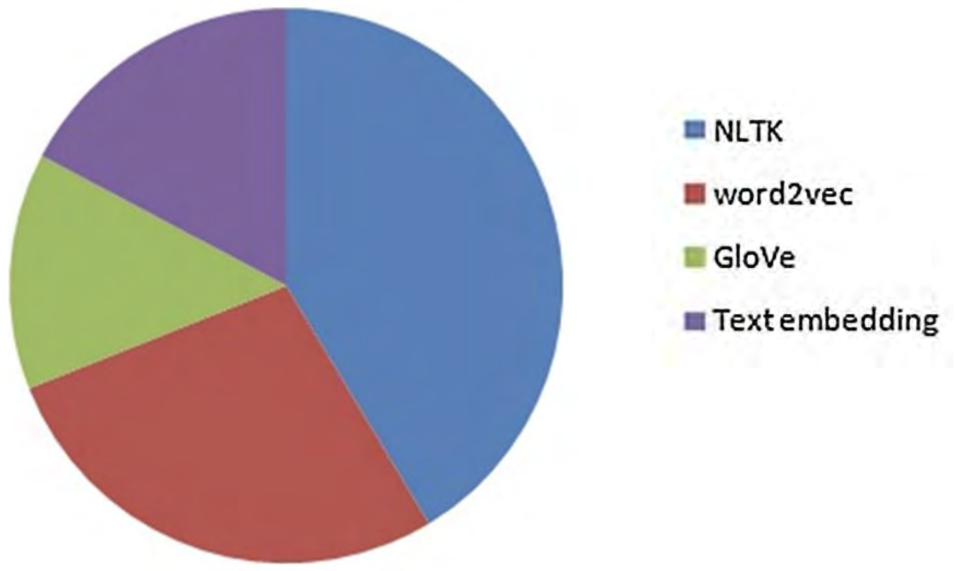
Intuition

- If you’re aware of what autoencoders are, you’ll find that the idea behind this network is similar to that of an autoencoder.
- You take an extremely large input vector, compress it down to a dense representation in the hidden layer, and then instead of reconstructing the original vector as in the case of autoencoders, you output probabilities associated with every word in the vocabulary.

Input/output

- Now the question arises, how do you input a single target word as a large Vector? The answer is One-Hot Encoding.
- Let’s say our vocabulary contains around 10,000 words and our current target word ‘fox’ is present somewhere in between. What we’ll do is, put a 1 in the position corresponding to the word ‘fox’ and 0 everywhere else, so we’ll have a 10,000-dimensional vector with a single 1 as the input.
- Similarly, the output coming out of our network will be a 10,000-dimensional vector as well, containing, for every word in our vocabulary, the probability of it being the context word for our input target word.

The NLTK is an NLP tool used to retrieve statistical features like POS, word count, sentence count, etc. With NLTK, we can miss the essay's semantic features. To find semantic features Word2Vec is the most used libraries to retrieve the semantic text from the essays. And in some systems, they directly trained the model with word embeddings to find the score.



Research questions

datasets

what are the datasets available for research on automated essay grading?

Question1

Features

what are the features extracted for the assessment of essays?

Question2

Evaluation Metrics

which are the evaluation metrics for measuring the accuracy of algorithms?

Question3

ML Techniques

What are the ML techniques used for AES, and how implemented?

Question4

challenges

What are the challenges and limitations in the current research?

Question5

ITIC systems

Q3-Evaluation Metrics

The majority of the AES systems are using three evaluation metrics:

quadrated weighted kappa (QWK)

The quadratic weighted kappa will find agreement between human evaluation score and system evaluation score and produces value ranging from 0 to 1.

$$\kappa = 1 - \frac{\sum_{i,j} \mathbf{W}_{i,j}\mathbf{O}_{i,j}}{\sum_{i,j} \mathbf{W}_{i,j}\mathbf{E}_{i,j}}$$

Mean Absolute Error (MAE)

The Mean Absolute Error is the actual difference between human-rated score to system-generated score. The mean square error (MSE) measures the average squares of the errors, i.e., the average squared difference between the human-rated and the system-generated scores. MSE will always give positive numbers only.

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n}$$

Pearson Correlation Coefficient (PCC)

Pearson’s Correlation Coefficient (PCC) finds the correlation coefficient between two variables. It will provide three values (0, 1, −1). "0" represents human-rated and system scores that are not related. "1" represents an increase in the two scores. "−1" illustrates a negative relationship between the two scores.

$$\rho_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y}$$

QWK

Quadratic weighted Kappa(QWK)

- The output of an AES system can be compared to the ratings assigned by human annotators using various measures of correlation or agreement. Quadratic weighted Kappa(QWK) as a famous measurement in this field is calculated as follows.
- First, a weight **matrix W** is constructed according to below equation where i and j are the reference rating (assigned by a human annotator) and the hypothesis rating (assigned by an AES system), respectively, and N is the number of possible ratings:

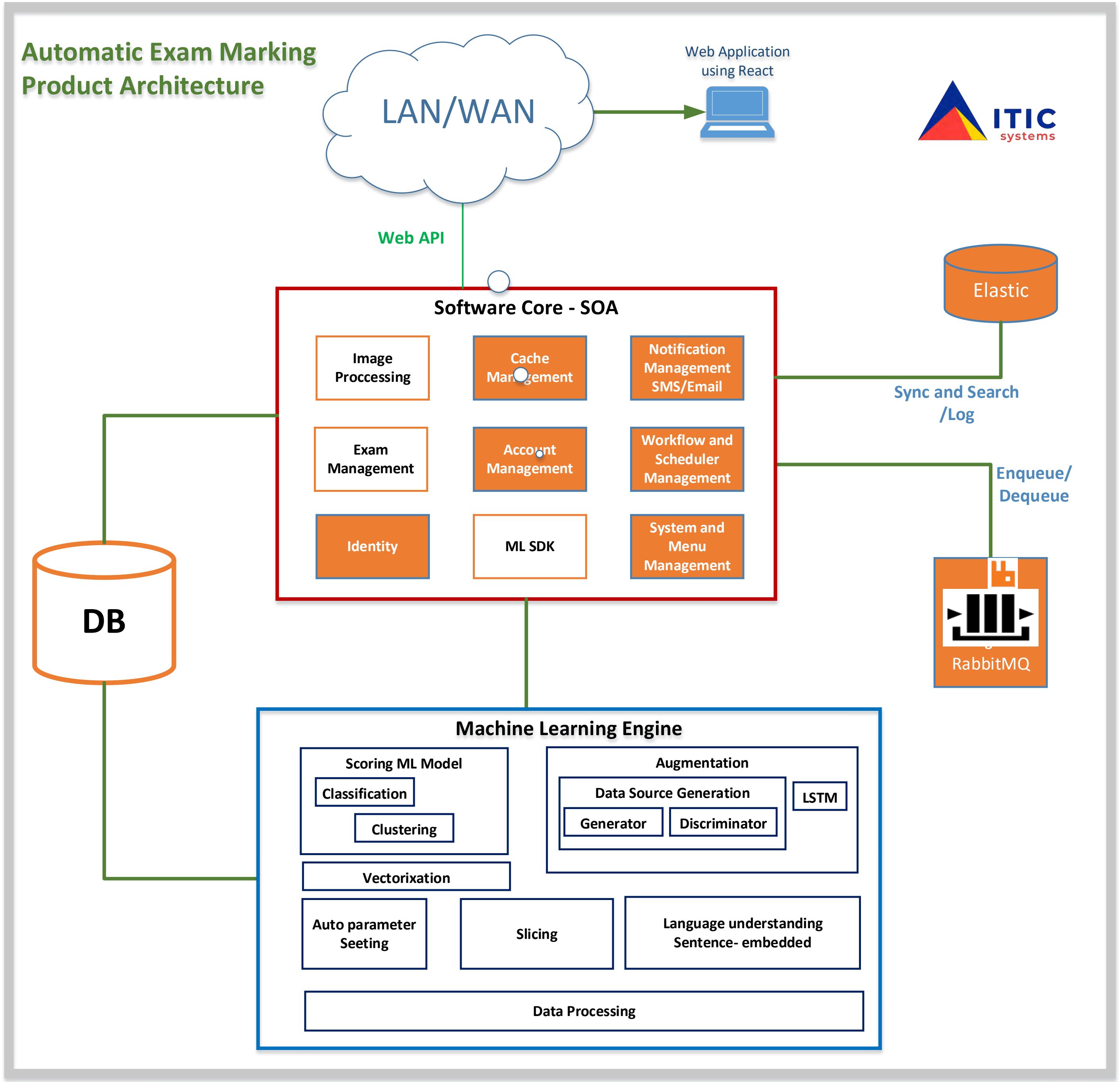
$$\mathbf{W}_{i,j} = \frac{(i-j)^2}{(N-1)^2}$$

- A **matrix O** is calculated such that $O_{i,j}$ denotes the number of essays that receive a rating i by the human annotator and a rating j by the AES system.
- An expected count **matrix E** is calculated as the outer product of histogram vectors of the two (reference and hypothesis) ratings. The matrix E is then normalized such that the sum of elements in E and the sum of elements in O are the same. Finally, given the matrices O and E, the QWK score is calculated according to:

$$\kappa = 1 - \frac{\sum_{i,j} \mathbf{W}_{i,j}\mathbf{O}_{i,j}}{\sum_{i,j} \mathbf{W}_{i,j}\mathbf{E}_{i,j}}$$

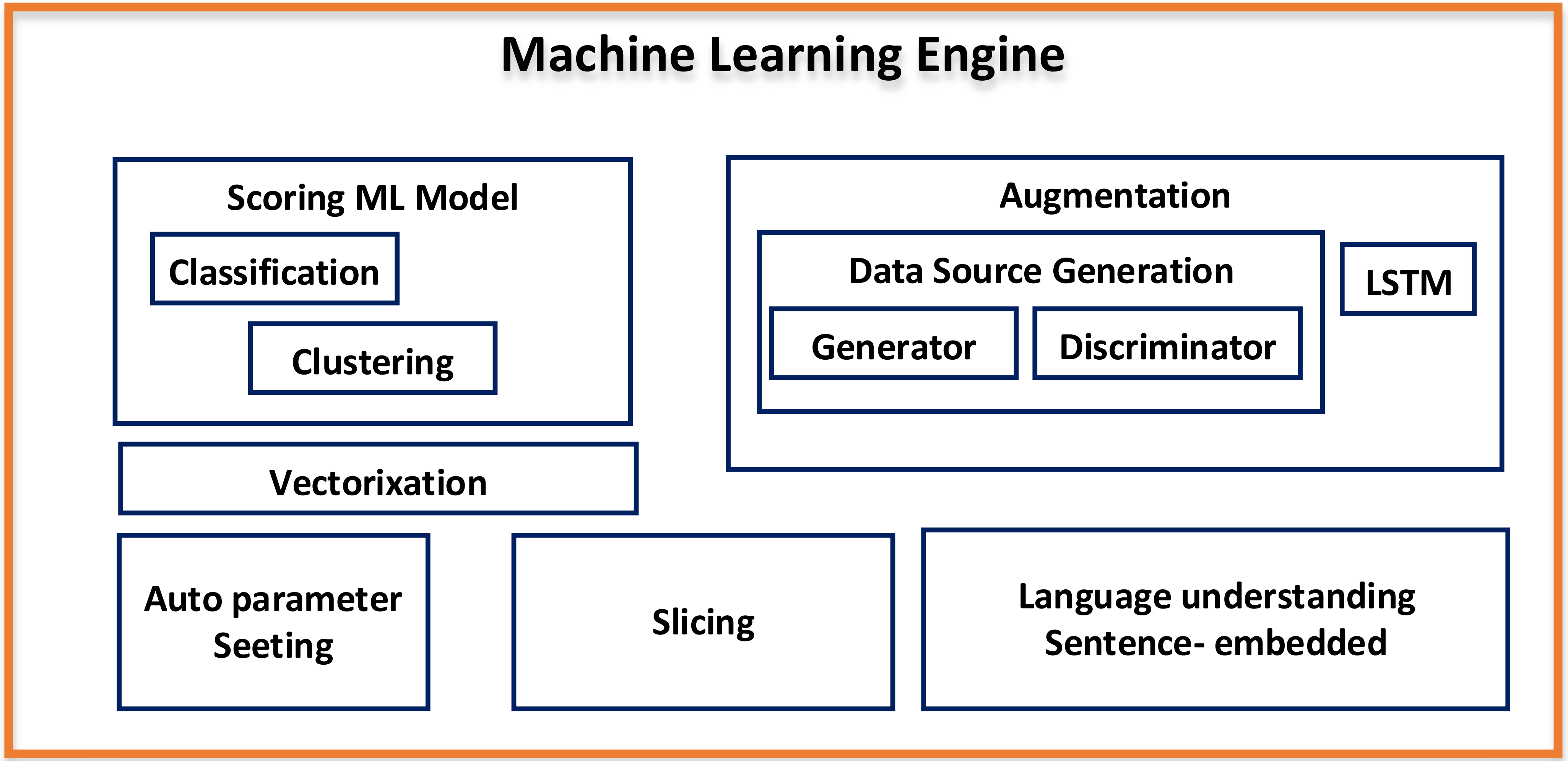
Research questions

datasets	Features	Evaluation Metrics	ML Techniques	challenges
what are the datasets available for research on automated essay grading?	what are the features extracted for the assessment of essays?	which are the evaluation metrics for measuring the accuracy of algorithms?	What are the ML techniques used for AES, and how implemented?	What are the challenges and limitations in the current research?
Question1	Question2	Question3	Question4	Question5



Product Architecture

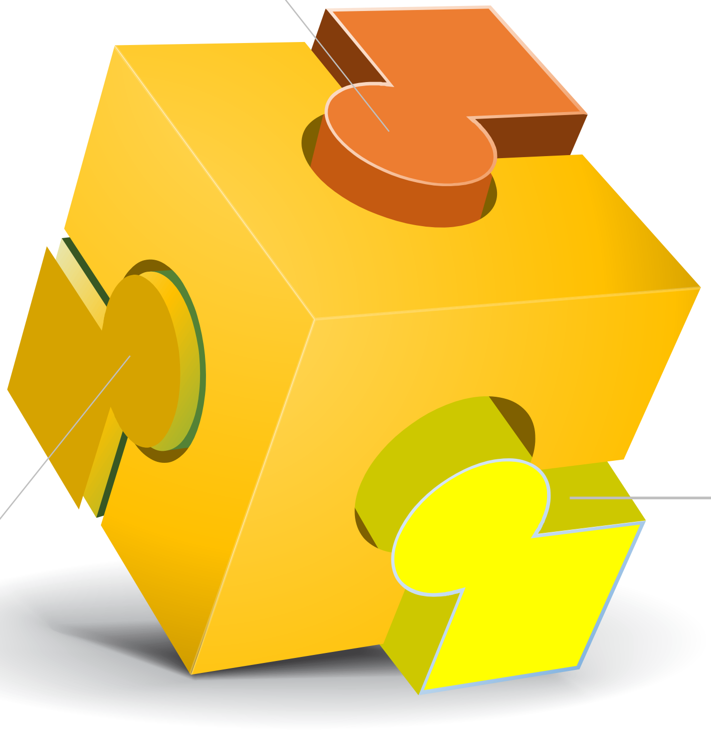
Sample Component of ML Engine



Q4-Machine Learning Techniques

Regression

regression



Neural Networks

Neural Networks

Classification

Classification

Research questions

datasets

what are the datasets available for research on automated essay grading?

Question1

Features

what are the features extracted for the assessment of essays?

Question2

Evaluation Metrics

which are the evaluation metrics for measuring the accuracy of algorithms?

Question3

ML Techniques

What are the ML techniques used for AES, and how implemented?

Question4

challenges

What are the challenges and limitations in the current research?

Question5

- The vast majority of essay scoring systems are dealing with the efficiency of the algorithm. But there are many challenges in automated essay grading systems. One should assess the essay by following parameters like:
- The relevance of the content to the prompt
- Development of ideas
- Cohesion: There is no discussion about the cohesion of the essays.
- Coherence: There is no discussion about the coherence of the essays.
- Domain knowledge: All researches concentrated on extracting the Features using some NLP libraries, trained their models, and testing the results.

Research questions

datasets

what are the datasets available for research on automated essay grading?

Features

what are the features extracted for the assessment of essays?

Evaluation Metrics

which are the evaluation metrics for measuring the accuracy of algorithms?

ML Techniques

What are the ML techniques used for AES, and how implemented?

challenges

What are the challenges and limitations in the current research?

ITIC systems

Data Augmentation

- high cost of collecting human-rated essays can be a bottleneck in building a cutting-edge scoring system. Automatic **data augmentation** can be a solution to the chronic problem of a lack of training data.
- EssayGAN is an automatic **essay generator** based on generative adversarial networks (GANs), which can automatically augment essays rated with a given score.
- Conventional GANs consist of two sub-networks: a **generator** that produces fake data and a **discriminator** that differentiates real from fake data. The core idea of GANs is to play a min–max game between the discriminator and the generator, i.e., adversarial training. The goal of the generator is to generate data that the discriminator believes to be real.

ITIC systems

EssayGAN Structure

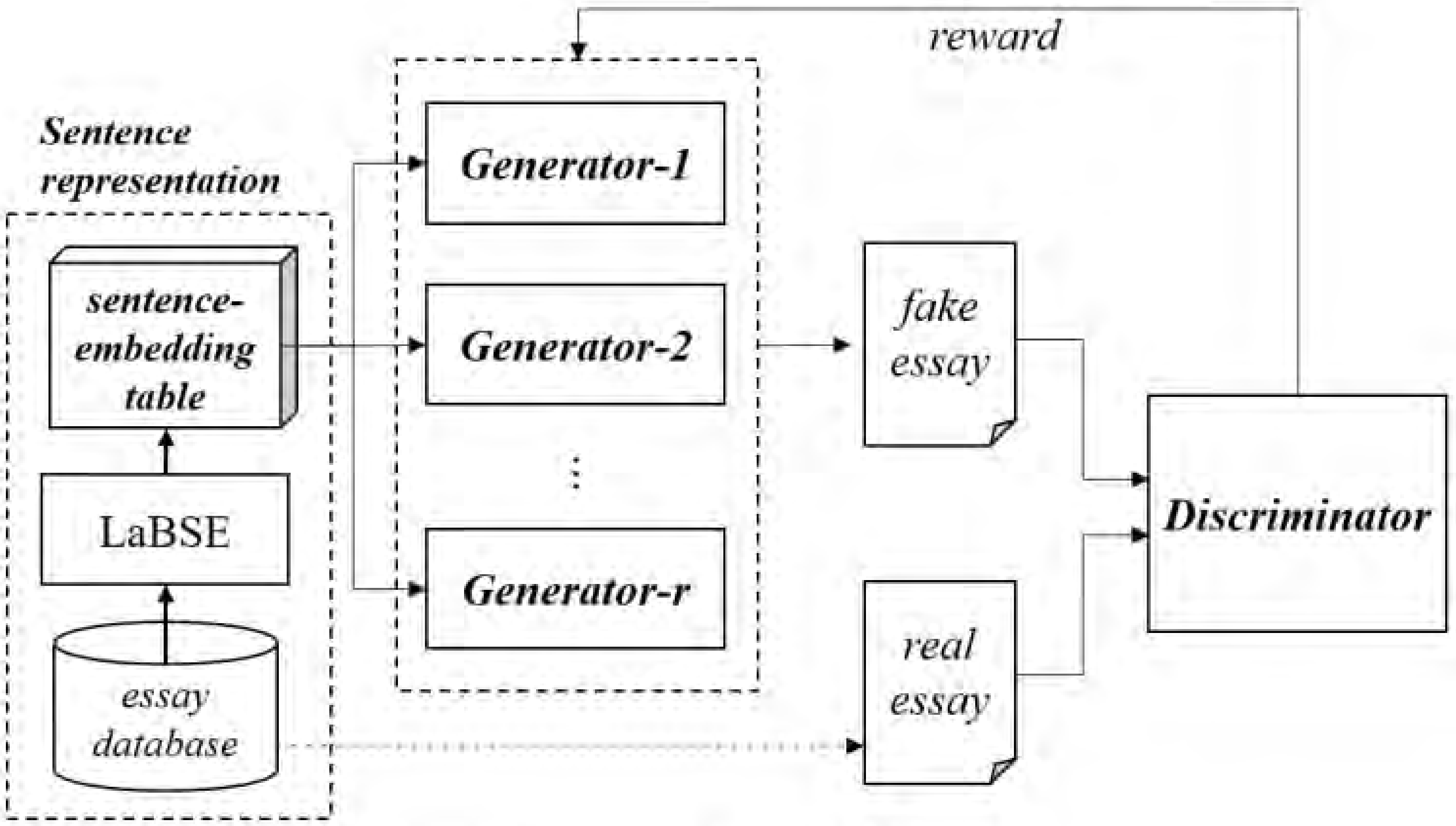


Figure 1. The overall architecture of EssayGAN.

Sentence Representation

- LaBSE is a pre-trained model built on a BERT-like architecture and uses the masked language model (MLM) and the translation language model (TLM). It is then fine-tuned using a translation ranking task. The resulting model can provide multilingual sentence embeddings in a single model.
- MLM is a language model trained to predict the missing words in a sentence based on the context provided by the surrounding words. This is done by masking some of the words in the input text and training the model to predict the masked words based on the context of the non-masked words.
- The sentence embeddings of all sentences are calculated in the training data, using LaBSE in advance, and then saved those embeddings in a sentence-embedding table. After that, the embeddings were exploited by the discriminator and the generators of EssayGAN.

BERT

- **BERT** is based on the transformer architecture. Specifically, BERT is composed of Transformer encoder layers.
- BERT was **pre-trained** simultaneously on two tasks:
- language modeling (15% of tokens were masked, and the training objective was to predict the original token given its context)
- next sentence prediction (the training objective was to classify if two spans of text appeared sequentially in the training corpus).
- As a result of this training process, BERT learns **latent representations** of words and sentences in context. After pre-training, BERT can be fine-tuned with fewer resources on smaller datasets to optimize its performance on specific tasks such as NLP tasks (language inference, text classification) and sequence-to-sequence based language generation tasks (question-answering, conversational response generation).
- The pre-training stage is significantly more computationally expensive than **fine-tuning**.

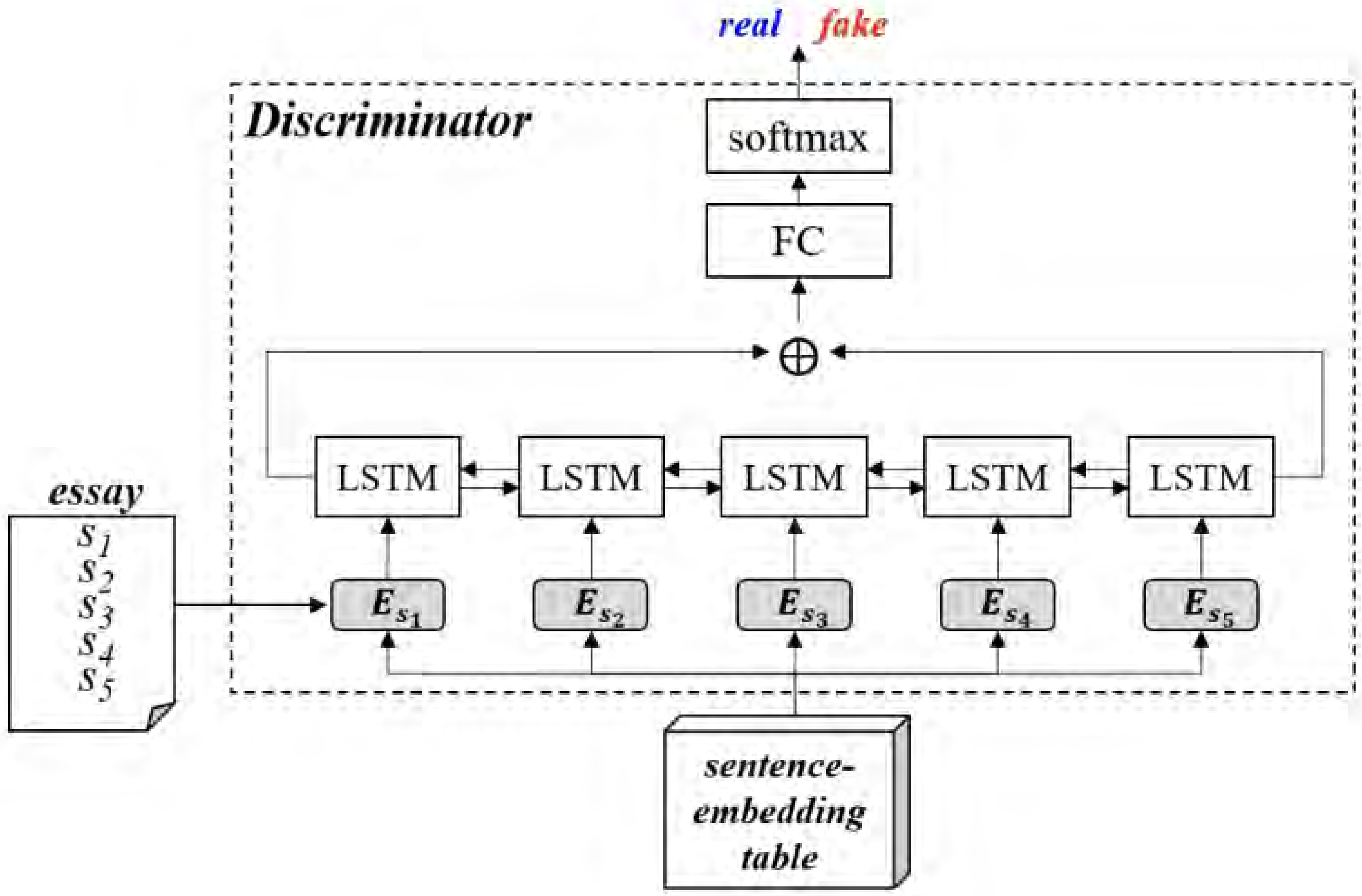


Figure 2. The architecture of the discriminator used in EssayGAN.

Discriminator

- The goal of the discriminator is to distinguish between human-written and generator composed essays. The discriminator is built based on **bi-directional LSTM networks**, as shown in Figure 2.
- The i-th sentence, s_i , in an input essay is converted into an embedding vector, E_{si}, by looking up the sentence-embedding table.
- Sentence embeddings are fed into the **LSTM** hidden states, and the first and the last hidden states are concatenated into a representation of the essay. The final layer of the discriminator **outputs a scalar value** indicating the **likelihood** that an input essay is **real**.
- The discriminator is trained to output a value as close as possible to 1 for real essays and as small as possible for fake essays. The output value of the discriminator is provided to the generators as a reward value.

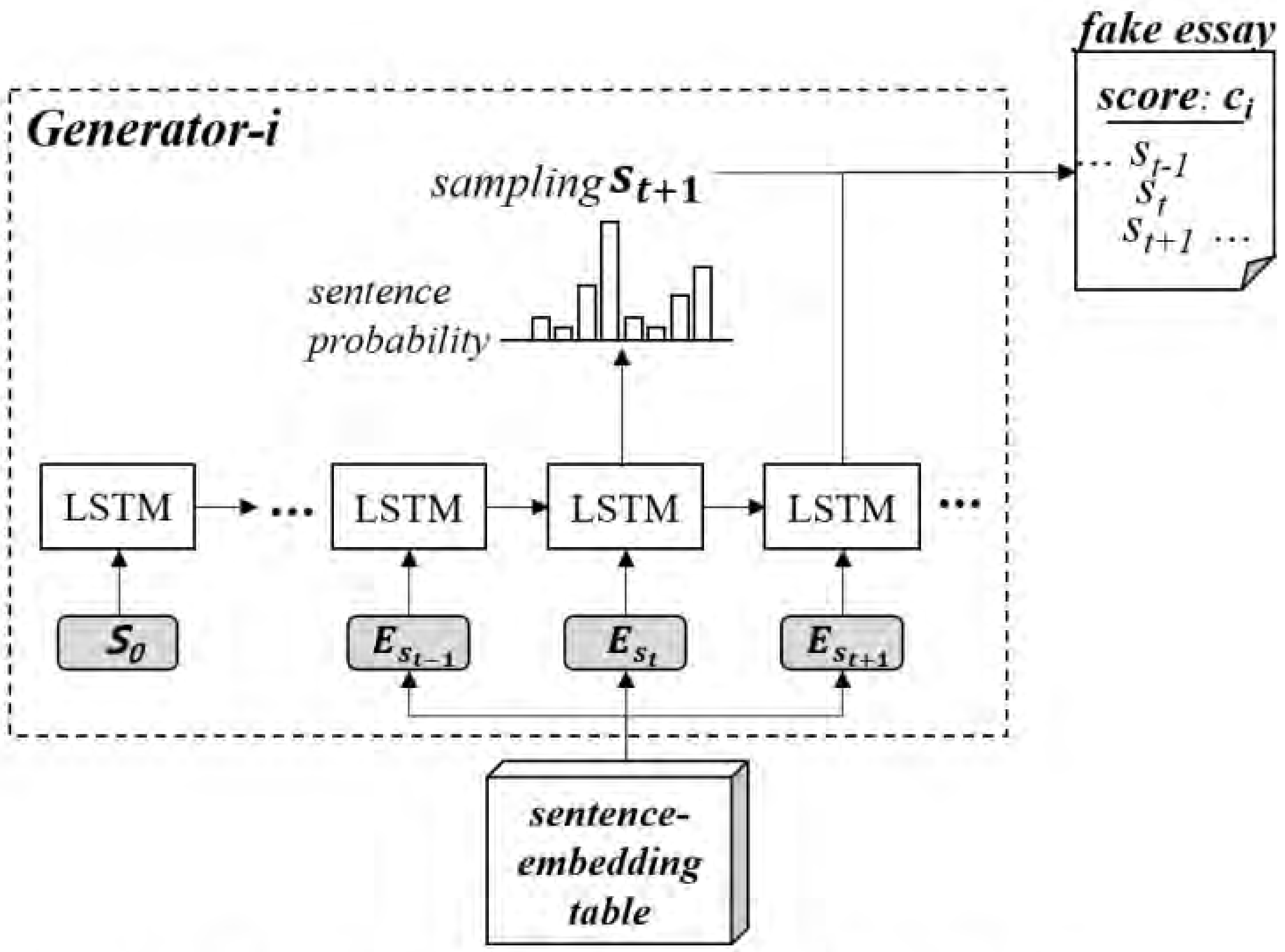
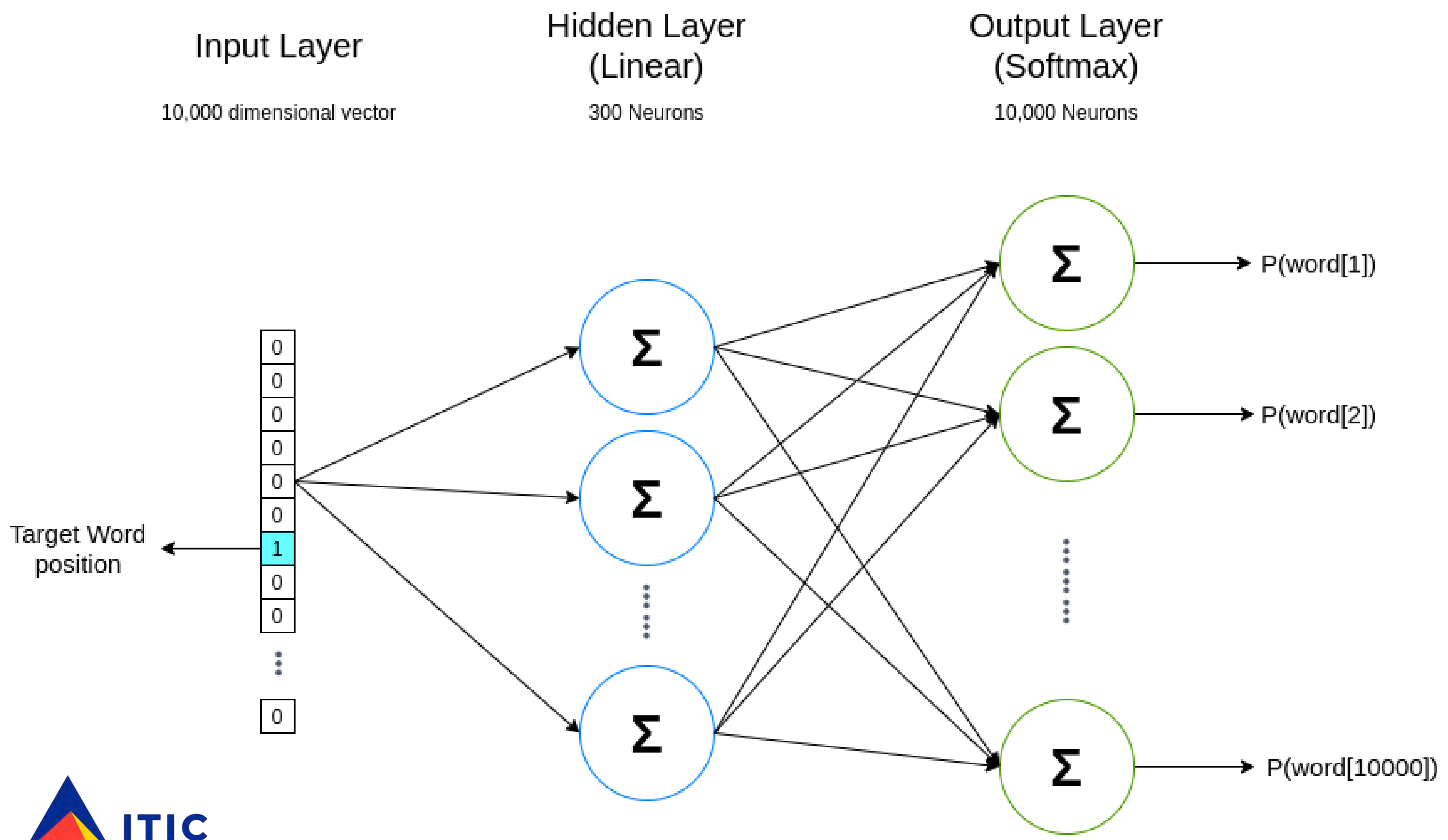


Figure 3. The architecture of the *i*-th generator in EssayGAN.

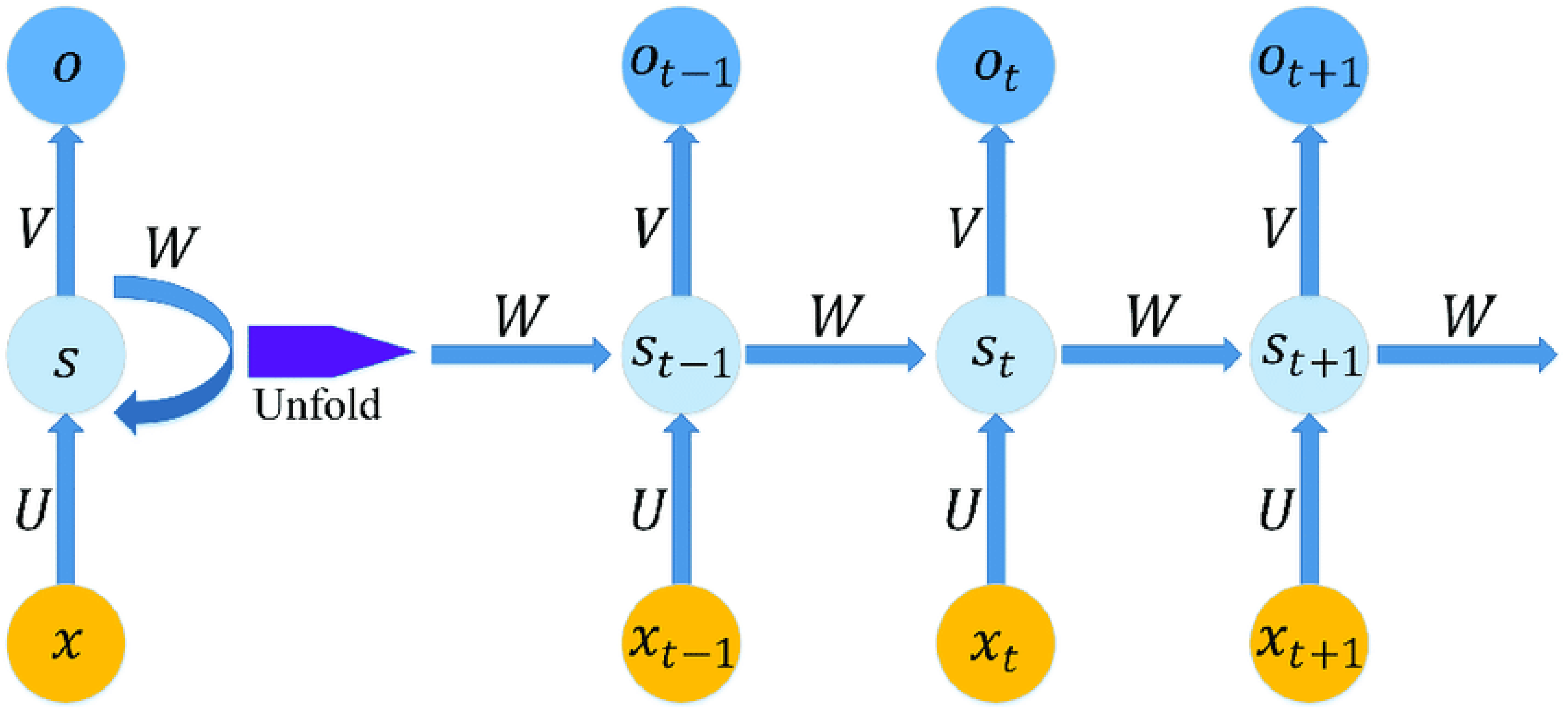
- Figure 3 depicts the architecture of the i-th generator assigned to generate essays scored as ci . We trained r generators to generate an essay with a specified score. The value of r is determined by the range of scores, which are specified in a scoring rubric.
- We utilized **LSTM** networks as a basic architecture of the generators. The LSTM networks were initially pre-trained with a sentence-level language model using a training dataset and using a conventional maximum likelihood estimation (MLE) method. Therefore, the pre-trained LSTM networks can predict the most likely next sentence based on previously selected sentences.
- After the pre-training phase, adversarial training was employed to train the generators and the discriminator in turn. The output layer of each LSTM cell has the same dimensions as a sentence-level one-hot vector that can identify a specific sentence.

neural network

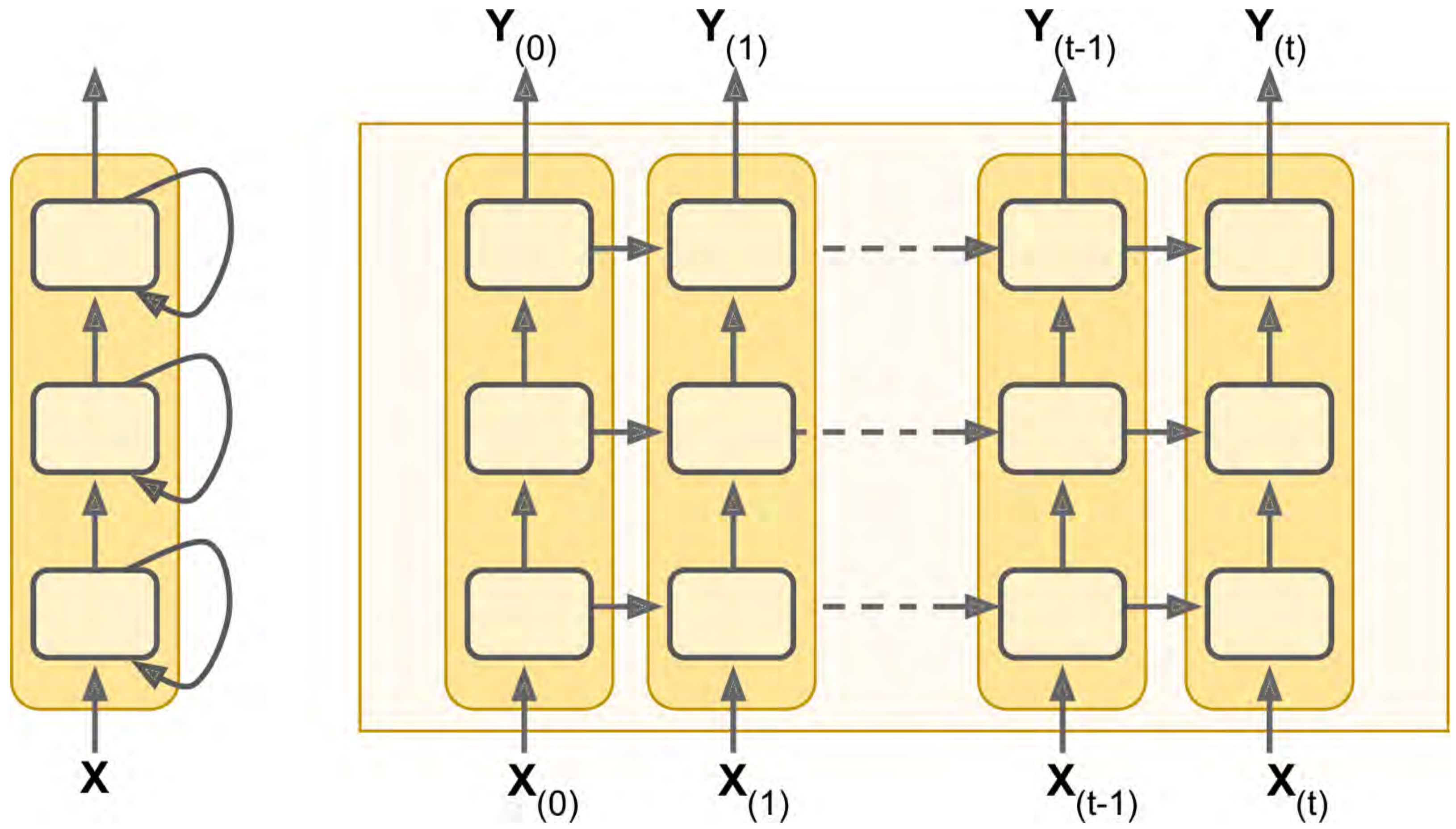


https://www.youtube.com/watch?v=ILsA4nyG7I0

recurrent neural network (RNN)



LSTM



ITIC systems