Towards partial fulfilment for Undergraduate Degree Level
Programme Bachelor of Technology in Computer Engineering

Final Stage Project Evaluation Report on:

Script Generating AI .

Prepared by:

Admission No.	Student Name
U18CO015	BHAUMIK SADRANI
U18CO049	HITESH PANDA
U18CO060	PRATEEK PRAVANJAN
U18CO094	NINAD SANJAY LAKADE

Class : B.TECH. IV (Computer Engineering)
8th Semester

Year : 2021-2022

Guided by : Prof. Devesh C Jinwala

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SARDAR VALLABHBHAI NATIONAL INSTITUTE OF TECHNOLOGY, SURAT - 395 007 (GUJARAT, INDIA)

Script-Generating AI

Bhaumik Sadrani, Hitesh Panda, Ninad Lakade, Prateek Pravanjan

May 13, 2022

Contents

List of Figures

Abstract

Natural Language has been an emerging field since the success of Image based Deep Learning, making great strides in enabling computers to understand and create natural language closer to humans. Humans adore stories. It provides the best meaning to their interpretation of everything they experience. NLP comes with the greatest challenge of understanding the context of natural language for which the model/models need to achieve awareness of the environment and have human common sense. The report here provides details of performance of various models and training optimisations.[no fucking idea but write something like comparing the performance of models and training optimizations].

Acknowledgements

We have taken efforts in this project. However, it would not have been possible without the kind support and help of the resources and opensource tools. We would like to extend our sincere thanks to all the maintainers who have put a lot into making this easier. They have been the giants, whose shoulders we have been standing on.

We are highly indebted to our guide Dr. Devesh C Jinwala for his guidance and constant supervision as well as for providing necessary information regarding the project and also for his support in completing the project. We would like to express our gratitude towards our parents for their kind co-operation and encouragement throughout completion of this project. We would like to express our special gratitude and thanks to Dr. Rupa G Mehta, our HOD, and SVNIT for providing us the opportunity to learn and grow as professionals and as individuals. Our thanks and appreciations also go to our colleagues and communities on the web who have willingly helped us out with their abilities.

Acronyms

ANN Artificial Neural Network

BPTT BackPropagation Through Time

CNN Convolutional Neural Network

CV Computer Vision

DL Deep Learning

LSTM Long Short Term Memory

LTM Long Term Memory

RNN Recurrent Neural Networks

STM Short Term Memory

1 Introduction

In the process of Computer Vision establishing Deep Learning (DL) as a paradigm with near-limitless potential the world, it attracted enough able-minds to build a proper foundation for itself. Meanwhile, despite Natural Language Generation and Understanding having been worked on a long time, the deep learning paradigm arrived relatively very late making room for the field to mature and inciting innovation and exploration in the domain.

Our focus is on text understanding and generation, to design an *efficient* language model for producing short text snippets for an incomplete script.

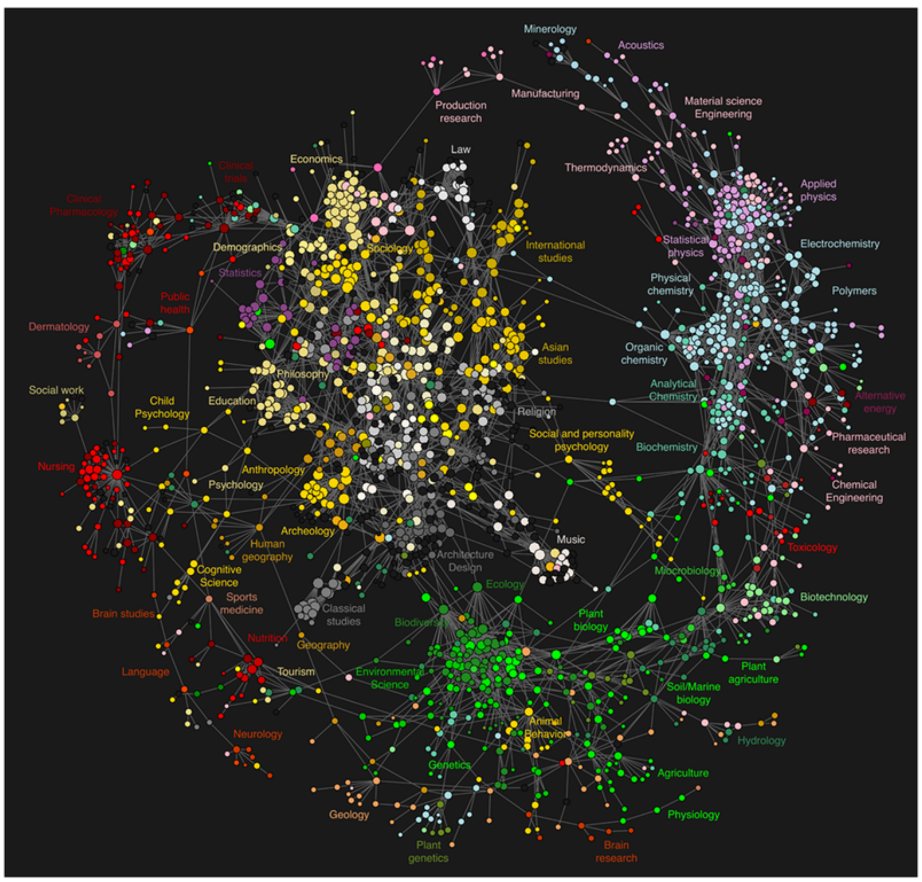

Figure 1: Knowledge Graph

Natural Language Understanding was initiated with creating specific rule sets. Taking english as the context in these explanations, the rule set will have groups(i.e, nouns, pronouns etc.) in which every word in the language can be classified to. Thereon, we can set phrase structure rules**??** and generate parse trees and knowledge graphs. This had it's limitations. Despite the obvious limitations, the primitive NLU approach kickstarted the birth and eventual dominance of Google Search in our lives. The advent of NLP invited us to unknown lands with infinite possibilites — all thanks to the universal function approximators we like to call Neural Nets — delivering some of the very well-known handy tools like Alexa and Siri.
Taking notes from previous iterations of the work we have focused more on optimizing[2] in a pythonic way to reduce overhead on training and using regularization methods.

1.1 Applications

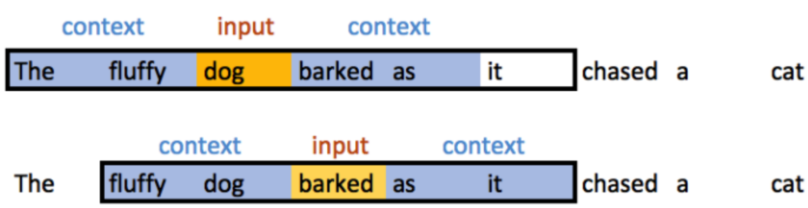

Figure 2: Applications

To be applied on scripts or completing novels or any other similar reading material.

1.2 Motivation

As NLG is a developing subject and a doorway to higher-order machine interaction, it has piqued our interest and curiosity. There are opportunities to develop unique methods and concepts in this environment, and given the widespread adoption of NLU on our mobile and IoT devices, we took upon this as inspiration for the project. Because present methodologies can be considerably improved, DL researchers are focusing their efforts on finding superior alternatives. In comparison to computer vision, NLP presents unique obstacles.

It's a golden ratio of tensor weights in Computer Vision (CV) that can generalise and discriminate what the pixels reveal. A basic tree-generation of the words (tokens) and organising them in a certain manner, as used by Neural Nets, would not serve for NLP.The NNs must learn the context, which is the most intriguing part. Understanding what natural languages(human languages) are can be a significant step in achieving AGI and soon enough an ASI.

2 Theoretical Background and Literature Survey

An ideal DL project requires many parts to work together. The processes carry on as data preparation, model training ,and deployment of which the latter is not in consideration for the project at hand. Once the context is understood by this mystical back-box, it can be of a wide range of use. In the current project, we will be implementing both NLU and NLG.

Sections below are a brief understanding of some of the frequently used concepts that will be required to understand, written with the notion of the reader having no a posteriori knowledge.

General Structure

As we saw previously, we need to carry out a number of steps to build a successful model.

The approach we choose has the following steps.

1. Data Processing
   (a) Get the data.
   (b) Explore the data.
   (c) Implement pre-processing functions.
2. Build the Neural Network
   (a) Input
   (b) Build RNN Cell
   (c) Embed Words
   (d) Build RNN
   (e) Build Neural Networks.
3. Neural Network Training.
4. Implement Generate Functions.
5. Get TV script.

General Architecture

Embedding Layer The model should take our word tokens and firstly pass it through our embedding layer. This layer will be responsible for converting out word tokens or integers into embeddings of specific size. These word embeddings are then fed to the next layer of Long Short Term Memory (LSTM) cells.

The main purpose of using embedding layer is dimensionality reduction.

Contiguous LSTM Layer Our LSTM layer is defined by hidden state size and number of layers. At each step, an LSTM cell will produce an output and a new hidden state. The hidden state will be passed to next cell as input (memory representation).

Final **Fully Connected Linear Layer** The output generated by LSTM cell will be then fed into a Sigmoid activated fully-connected linear layer. This layer is responsible for mapping LSTM output to desired output size.

The output of the sigmoid function will be the probability distribution of most likely next word.

2.1 Tokenization

The initial task in processing unstructured data is tokenization. The texts used for learning will be taken as tokens from the sentences put in the NN. Then the further steps are carried.

2.2 Stemming and Lemmetization

Stemming helps congregate words that mean the same but are just of different forms due to the tense or the nature of its use. To give an example, running, ran and run carry the same meaning and therefore will be stemmed together. To beat the limitation of stemming, lemmetization is invited. It doesn't make the use of stemming obsolete, rather works in tandem. Lemmetization learns the meaning through the dictionary definition of the token and derives the root of the token. An instance would be universal and university, which could have been stemmed together.

2.3 Data preparation for NLG

Considering a plethora of vectors to choose from, in order to provide data, the model trims down the details as per the requirement. This will be the step commencing the NLG model. It is called as Content Determination.

2.4 Building Natural Language from structured data

Data is first interpreted and patterns are recognized as learnt by NNs. This data is put to context, which will be put in narrative structure. The next process to be carried is quintessential to the reason behind NLG. The correlation between the sentences is learnt and an appropriate arrangement of the sentences is created, so it makes sense to the end user.

A sanity check is then carried out, wherein, the grammar is checked. After this validation, the data is put into templates that output in the right format finally to be presented to the user.

Some of the frequently used pre-training architecures would be BERT, word2vec, GloVe and Subword Embedding. The existing text sequences of the large corpora are used to build the necessary connections, so we don't have to put labels ourselves. This would be also known as *self-supervised learning*. The pretrained corpora is now fed into an RNN, CNN, MLP or Attention Model. The model trains itself with the data and tries to achieve minimum loss in the predictions.

2.5 Dropouts

Dropouts is a regularisation technique which helps to avoid *overfitting*. Switching off some neurons at each training step is termed as dropout. Mathematically, each neuron has some probability P of being ignored, which is the dropout rate. This dropout rate depends on various parameters like the network type, layer size and degree of overfitting. The information spread is more even across the network, which leads to generalising the model as it becomes less sensitive to input changes. Dropouts are only used in training and not while predicting generated text.

Networks with neurons dropped out can be treated as *Monte Carlo samples*. This provides a mathematical basis to give reason about the models uncertainty and this in turn helps the model improve its performance. It is implemented by applying dropouts at the testing time. Hence we can obtain many predictions, one for each model for the distribution analysis. An advantage of this is that there is no requirement to change the architecture of model. This kind of dropout can also be used on a model that has been previously trained.

2.6 Early stopping

Early stopping is a regularisation technique which can help to prevent overfitting when training with an iterative model. By tuning the parameters, the model chases the loss function on the training data. Another set of data is kept as the validation set and the loss function is recorded for this validation data. The training stops when there is no observable improvement on the validation set. This method of stopping early based on the validation set performance is called Early Stopping. An advantage of early stopping is that it requires lesser number of epochs to train.

2.7 Internal Covariate Shift and Batch Normalisation

Internal Covariate Shift refers to the change in distribution of network activations due to change in network parameters during training. The deeper the network the more disorder Internal Covariate Shift can cause. Neural Networks learn to adjust their weights according to a mathematical set of rules as more the layers, more the complexity. The goal is to achieve stability and improve the connection between the output layer's results and each node of the hidden layer.

Batch Normalisation Scaling data is a key aspect to building any model. Here it is made sure that data is not only scaled before training but also remains to be scaled while training takes place. This is done using a batch normalising transform[1]. These operations involve standardization, normalization, rescaling and shifting of offset of input values coming into the BN layer.

$$y = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} \cdot \gamma + \beta$$

2.8 One Hot Encoding VS Word2Vec

Converting data in order to prepare it for an algorithm and thus get a better prediction is known as One Hot Encoding. Using this method, each categorical value is converted into a categorical column. A binary value is assigned to it and each integer is represented as a binary vector. The issue arises when there is high cardinality. The vector space finds it difficult to accommodate higher dimensions. Integer encoding is usually not enough for categorical variables where no ordinal relationship exist. Using this encoding and allowing the model to assume a natural ordering between categories may result in poor performance or unexpected results.

word2vec on the other hand retains the semantic meaning of different words in a document. The context information is not lost. The size of the embedding vector is very small in word2Vec, hence each dimension in the embedding vector contains information about one aspect of the word. The requirement of sparse vectors is not necessary.

2.9 Recurrent Neural Networks

Recurrent Neural Networks[7] are used for sequential data tailored for ordinal or tailored problems like language translation, natural language processing, speech recognition and image captioning. They stand out from standard neural networks like Artificial Neural Network (ANN) and Convolutional Neural Network (CNN) for taking consideration of prior inputs to influence the current input and output. These standard neural networks see no learning on a priori basis and only train on currently fed data.

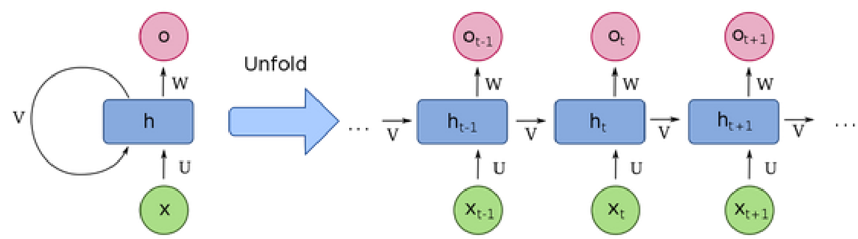

Figure 3: RNNs

Another trait of Recurrent Neural Networks (RNN) is the passing of parameters across the layers of the network. They share the same weight parameter within each layer of the network. They leverage on the BackPropagation Through Time (BPTT) algorithm[6] — a slightly different algorithm than backpropagation — to determine the gradients from calculating errors from input layer to the output layer which are summed every time step. The BPTT algorithm can be represented as

$$\frac{\delta E_N}{\delta W_x} = \sum_{i=1}^{N} \frac{\delta E_N}{\delta \bar{y}_N} \cdot \frac{\delta \bar{y}_N}{\delta \bar{s}_i} \cdot \frac{\bar{s}_i}{\delta W_x}$$

ShortComings RNNs have some drawbacks despite these mentioned advantages. One such is the VANISHING GRADIENTS PROBLEM[3]. This happens during backpropagation, wherein the gradients of the cells that carry information from the start of a sequence goes through matrix multiplications by small numbers and reach close to 0 in long sequences — information towards the beginning of the sequence has almost no effect at the end of the sequence. The other shortcoming, namely EXPLODING GRADIENTS PROBLEM[3], wherein the gradients become too large creating an unstable model resulting in too large numbers. One way to avoid such possibilities is to reduce the number of hidden layers and gradient clipping can mitigate the exploding gradients problem.

2.10 Long-Short Term Memory

LSTM[7] is a popular variant of the RNN architecture and like an RNN it works on sequences of data *viz.* text generation, video classification, music generation *etc.* LSTM fills in for the drawbacks of RNN like the vanishing gradients problem by having a memory gating mechanism which allows long term memory to continue flowing into the LSTM cells — 100s of elements in a sequence. LSTM has two components namely *Long Term Memory* and *Short Term Memory* the combined output of which update each other. LSTMs comprise of four gates.

Forget Gate Forget gate factors in the Long Term Memory (LTM) and decides which part to keep and which to forget. LTM gets multiplied by a forget factor in order to forget unwanted parts of LTM.

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

Learn Gate This considers the *short term memory and event*, and partially retains information. The Short Term Memory (STM) and Event are combined using an activation function *(tanh)*, which we further multiply by a ignore factor

Remember Gate Remember gate receives the LTM coming from Forget gate and STM coming from Learn gate and combines them. Mathematically, remember gate adds LTM and STM.

Use Gate Picks the useful information from LTM and STM and generates a new STM.

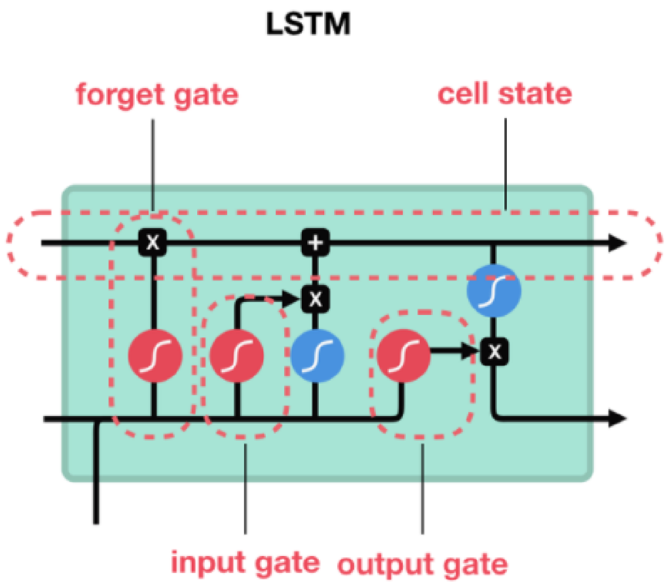

Figure 4: LSTM

3 Experimental Methodologies

3.1 RNN-LSTM Model

```
class LSTM_LSTM_Model(nn.Module):
    def __init__(self, dataset):
        super(LSTM_LSTM_Model, self).__init__()
        self.lstm_size = 128
        self.embedding_dim = 128
        self.num_layers = 3

        n_vocab = len(dataset.uniq_words)
        self.embedding = nn.Embedding(
            num_embeddings=n_vocab,
            embedding_dim=self.embedding_dim,
        )
        self.lstm = nn.LSTM(
            input_size=self.lstm_size,
            hidden_size=self.lstm_size,
            num_layers=self.num_layers,
            dropout=0.2,
        )

        self.fc = nn.Linear(self.lstm_size, n_vocab)
```

```python
    def forward(self, x, prev_state):
        embed = self.embedding(x)
        output, state = self.lstm(embed, prev_state)
        output, state = self.lstm(embed, prev_state)
        logits = self.fc(output)
        return logits, state

    def init_state(self, sequence_length):
        return (torch.zeros(self.num_layers, sequence_length, self.
                                            lstm_size),
                torch.zeros(self.num_layers, sequence_length, self.
                                                    lstm_size))
```

3.2 Batch Normalisation

```python
def forward(self, x, prev_state):
        embed = self.embedding(x)
        output, state = self.lstm(embed, prev_state)
        output = nn.BatchNorm2d(output.size(dim=1))
        logits = self.fc(output)
        return logits, state
```

3.3 Extracting and Transforming

```
class Dataset(torch.utils.data.Dataset):
    def __init__(self,args,dataset="reddit_dataset/reddit-
                                      cleanjokes.csv"):
        self.args = args
        self.dataset = dataset
        self.words = self.load_words()
        self.uniq_words = self.get_uniq_words()


        self.index_to_word = {index: word for index, word in
                                  enumerate(self.uniq_words
                                  )}
        self.word_to_index = {word: index for index, word in
                                  enumerate(self.uniq_words
                                  )}

        self.words_indexes = [self.word_to_index[w] for w in self.
                                  words]


    def load_words(self):
        """loads the dataset"""
        train_df = pd.read_csv(self.dataset)
        text = train_df['Joke'].str.cat(sep=' ')
        return text.split(' ')

    def get_uniq_words(self):
        word_counts = Counter(self.words)
        return sorted(word_counts, key=word_counts.get, reverse=
                                  True)

    def __len__(self):
        return len(self.words_indexes) - self.args.sequence_length

    def __getitem__(self, index):
        return (
            torch.tensor(self.words_indexes[index:index+self.args.
                                      sequence_length]),
            torch.tensor(self.words_indexes[index+1:index+self.args
                                      .sequence_length+1])
        )
```

3.4 Loading and Training the model

```
class LSTM_Model(nn.Module):
    def __init__(self, dataset):
        super(LSTM_Model, self).__init__()
        self.lstm_size = 128
        self.embedding_dim = 128
        self.num_layers = 3

        n_vocab = len(dataset.uniq_words)
        self.embedding = nn.Embedding(
            num_embeddings=n_vocab,
            embedding_dim=self.embedding_dim,
        )
        self.lstm = nn.LSTM(
            input_size=self.lstm_size,
            hidden_size=self.lstm_size,
            num_layers=self.num_layers,
            dropout=0.2,
        )
        self.fc = nn.Linear(self.lstm_size, n_vocab)

    def forward(self, x, prev_state):
        embed = self.embedding(x)
        output, state = self.lstm(embed, prev_state)
        # output = nn.BatchNorm2d(output.size(dim=1))
        logits = self.fc(output)
        return logits, state

    def init_state(self, sequence_length):
        return (torch.zeros(self.num_layers, sequence_length, self.
                                                lstm_size),
                torch.zeros(self.num_layers, sequence_length, self.
                                                        lstm_size))
```

```
class LSTM_RNN_Model(nn.Module):
    def __init__(self, dataset):
        super(LSTM_RNN_Model, self).__init__()
        self.lstm_size = 128
        self.embedding_dim = 128
        self.num_layers = 4

        n_vocab = len(dataset.uniq_words)
        self.embedding = nn.Embedding(
            num_embeddings=n_vocab,
            embedding_dim=self.embedding_dim,
        )

        self.lstm = nn.LSTM(
            input_size=self.lstm_size,
            hidden_size=self.lstm_size,
            num_layers=self.num_layers,
            dropout=0.2,
        )

        self.rnn = nn.RNN(
            input_size=self.lstm_size,
```

```
            hidden_size=self.lstm_size,
            num_layers=self.num_layers,
            dropout=0.2,
        )

        self.fc = nn.Linear(self.lstm_size, n_vocab)

    def forward(self, x, prev_state):
        embed = self.embedding(x)
        output, state = self.rnn(embed, prev_state)
        output, state = self.lstm(embed, prev_state)
        logits = self.fc(output)
        return logits, state

    def init_state(self, sequence_length):
        return (torch.zeros(self.num_layers, sequence_length, self.
                                    lstm_size),
                torch.zeros(self.num_layers, sequence_length, self.
                                    lstm_size))
```

4 Results

4.1 Trainng Epoch

These Batches we generated along with loss.

```
Epoch 0 Batch    0/13   train-loss = 8.822
Epoch 0 Batch    6/13   train-loss = 6.517
Epoch 0 Batch    12/13   train-loss = 6.606
Epoch 1 Batch    5/13   train-loss = 6.040
Epoch 1 Batch    11/13   train-loss = 6.165
Epoch 2 Batch    4/13   train-loss = 6.081
Epoch 2 Batch    10/13   train-loss = 6.049
Epoch 3 Batch    3/13   train-loss = 5.960
Epoch 3 Batch    9/13   train-loss = 5.886
Epoch 4 Batch    2/13   train-loss = 5.771
Epoch 4 Batch    8/13   train-loss = 5.752
Epoch 5 Batch    1/13   train-loss = 5.783
Epoch 5 Batch    7/13   train-loss = 5.766
Epoch 6 Batch    0/13   train-loss = 5.674
Epoch 6 Batch    6/13   train-loss = 5.676
Epoch 6 Batch    12/13   train-loss = 5.643
Epoch 7 Batch    5/13   train-loss = 5.492
Epoch 7 Batch    11/13   train-loss = 5.465
Epoch 8 Batch    4/13   train-loss = 5.388
Epoch 8 Batch    10/13   train-loss = 5.393
Epoch 9 Batch    3/13   train-loss = 5.302
Epoch 9 Batch    9/13   train-loss = 5.233
Epoch 10 Batch    2/13   train-loss = 5.106
Epoch 10 Batch    8/13   train-loss = 5.077
Epoch 11 Batch    1/13   train-loss = 5.055
Epoch 11 Batch    7/13   train-loss = 5.089
Epoch 12 Batch    0/13   train-loss = 4.977
Epoch 12 Batch    6/13   train-loss = 4.983
Epoch 12 Batch    12/13   train-loss = 5.021
Epoch 13 Batch    5/13   train-loss = 4.876
Epoch 13 Batch    11/13   train-loss = 4.853
Epoch 14 Batch    4/13   train-loss = 4.814
Epoch 14 Batch    10/13   train-loss = 4.781
Epoch 15 Batch    3/13   train-loss = 4.770
Epoch 15 Batch    9/13   train-loss = 4.656
Epoch 16 Batch    2/13   train-loss = 4.575
Epoch 16 Batch    8/13   train-loss = 4.546
Epoch 17 Batch    1/13   train-loss = 4.522
Epoch 17 Batch    7/13   train-loss = 4.547
```

```
Epoch 18 Batch   0/13   train-loss = 4.465
Epoch 18 Batch   6/13   train-loss = 4.492
Epoch 18 Batch   12/13   train-loss = 4.513
Epoch 19 Batch   5/13   train-loss = 4.371
Epoch 19 Batch   11/13   train-loss = 4.324
Epoch 20 Batch   4/13   train-loss = 4.282
Epoch 20 Batch   10/13   train-loss = 4.268
Epoch 21 Batch   3/13   train-loss = 4.259
Epoch 21 Batch   9/13   train-loss = 4.195
Epoch 22 Batch   2/13   train-loss = 4.132
Epoch 22 Batch   8/13   train-loss = 4.084
Epoch 23 Batch   1/13   train-loss = 4.028
Epoch 23 Batch   7/13   train-loss = 4.087
Epoch 24 Batch   0/13   train-loss = 3.977
Epoch 24 Batch   6/13   train-loss = 4.025
Epoch 24 Batch   12/13   train-loss = 4.034
Epoch 25 Batch   5/13   train-loss = 3.933
Epoch 25 Batch   11/13   train-loss = 3.878
Epoch 26 Batch   4/13   train-loss = 3.824
Epoch 26 Batch   10/13   train-loss = 3.805
Epoch 27 Batch   3/13   train-loss = 3.817
Epoch 27 Batch   9/13   train-loss = 3.724
Epoch 28 Batch   2/13   train-loss = 3.672
Epoch 28 Batch   8/13   train-loss = 3.605
Epoch 29 Batch   1/13   train-loss = 3.566
Epoch 29 Batch   7/13   train-loss = 3.571
Epoch 30 Batch   0/13   train-loss = 3.529
Epoch 30 Batch   6/13   train-loss = 3.522
Epoch 30 Batch   12/13   train-loss = 3.535
Epoch 31 Batch   5/13   train-loss = 3.451
Epoch 31 Batch   11/13   train-loss = 3.376
Epoch 32 Batch   4/13   train-loss = 3.389
Epoch 32 Batch   10/13   train-loss = 3.318
Epoch 33 Batch   3/13   train-loss = 3.330
Epoch 33 Batch   9/13   train-loss = 3.217
Epoch 34 Batch   2/13   train-loss = 3.265
Epoch 34 Batch   8/13   train-loss = 3.169
Epoch 35 Batch   1/13   train-loss = 3.100
Epoch 35 Batch   7/13   train-loss = 3.099
Epoch 36 Batch   0/13   train-loss = 3.040
Epoch 36 Batch   6/13   train-loss = 3.058
Epoch 36 Batch   12/13   train-loss = 3.078
Epoch 37 Batch   5/13   train-loss = 3.037
Epoch 37 Batch   11/13   train-loss = 2.959
Epoch 38 Batch   4/13   train-loss = 2.929
Epoch 38 Batch   10/13   train-loss = 2.887
```

```
Epoch 39 Batch   3/13   train-loss = 2.894
Epoch 39 Batch   9/13   train-loss = 2.813
Epoch 40 Batch   2/13   train-loss = 2.882
Epoch 40 Batch   8/13   train-loss = 2.782
Epoch 41 Batch   1/13   train-loss = 2.715
Epoch 41 Batch   7/13   train-loss = 2.675
Epoch 42 Batch   0/13   train-loss = 2.675
Epoch 42 Batch   6/13   train-loss = 2.668
Epoch 42 Batch   12/13   train-loss = 2.713
Epoch 43 Batch   5/13   train-loss = 2.662
Epoch 43 Batch   11/13   train-loss = 2.602
Epoch 44 Batch   4/13   train-loss = 2.574
Epoch 44 Batch   10/13   train-loss = 2.566
Epoch 45 Batch   3/13   train-loss = 2.602
Epoch 45 Batch   9/13   train-loss = 2.508
Epoch 46 Batch   2/13   train-loss = 2.613
Epoch 46 Batch   8/13   train-loss = 2.499
Epoch 47 Batch   1/13   train-loss = 2.451
Epoch 47 Batch   7/13   train-loss = 2.403
Epoch 48 Batch   0/13   train-loss = 2.402
Epoch 48 Batch   6/13   train-loss = 2.427
Epoch 48 Batch   12/13   train-loss = 2.420
Epoch 49 Batch   5/13   train-loss = 2.414
Epoch 49 Batch   11/13   train-loss = 2.348
Epoch 50 Batch   4/13   train-loss = 2.352
Epoch 50 Batch   10/13   train-loss = 2.357
Epoch 51 Batch   3/13   train-loss = 2.381
Epoch 51 Batch   9/13   train-loss = 2.265
Epoch 52 Batch   2/13   train-loss = 2.379
Epoch 52 Batch   8/13   train-loss = 2.305
Epoch 53 Batch   1/13   train-loss = 2.283
Epoch 53 Batch   7/13   train-loss = 2.255
Epoch 54 Batch   0/13   train-loss = 2.270
Epoch 54 Batch   6/13   train-loss = 2.281
Epoch 54 Batch   12/13   train-loss = 2.224
Epoch 55 Batch   5/13   train-loss = 2.253
Epoch 55 Batch   11/13   train-loss = 2.215
Epoch 56 Batch   4/13   train-loss = 2.224
Epoch 56 Batch   10/13   train-loss = 2.163
Epoch 57 Batch   3/13   train-loss = 2.213
Epoch 57 Batch   9/13   train-loss = 2.081
Epoch 58 Batch   2/13   train-loss = 2.175
Epoch 58 Batch   8/13   train-loss = 2.090
Epoch 59 Batch   1/13   train-loss = 2.043
Epoch 59 Batch   7/13   train-loss = 1.971
Epoch 60 Batch   0/13   train-loss = 1.992
```

```
Epoch 60 Batch   6/13   train-loss = 1.993
Epoch 60 Batch   12/13   train-loss = 2.009
Epoch 61 Batch   5/13   train-loss = 2.039
Epoch 61 Batch   11/13   train-loss = 1.963
Epoch 62 Batch   4/13   train-loss = 1.951
Epoch 62 Batch   10/13   train-loss = 1.911
Epoch 63 Batch   3/13   train-loss = 1.988
Epoch 63 Batch   9/13   train-loss = 1.900
Epoch 64 Batch   2/13   train-loss = 2.002
Epoch 64 Batch   8/13   train-loss = 1.891
Epoch 65 Batch   1/13   train-loss = 1.844
Epoch 65 Batch   7/13   train-loss = 1.773
Epoch 66 Batch   0/13   train-loss = 1.835
Epoch 66 Batch   6/13   train-loss = 1.848
Epoch 66 Batch   12/13   train-loss = 1.826
Epoch 67 Batch   5/13   train-loss = 1.867
Epoch 67 Batch   11/13   train-loss = 1.793
Epoch 68 Batch   4/13   train-loss = 1.779
Epoch 68 Batch   10/13   train-loss = 1.777
Epoch 69 Batch   3/13   train-loss = 1.843
Epoch 69 Batch   9/13   train-loss = 1.782
Epoch 70 Batch   2/13   train-loss = 1.849
Epoch 70 Batch   8/13   train-loss = 1.781
Epoch 71 Batch   1/13   train-loss = 1.731
Epoch 71 Batch   7/13   train-loss = 1.708
Epoch 72 Batch   0/13   train-loss = 1.717
Epoch 72 Batch   6/13   train-loss = 1.733
Epoch 72 Batch   12/13   train-loss = 1.716
Epoch 73 Batch   5/13   train-loss = 1.787
Epoch 73 Batch   11/13   train-loss = 1.723
Epoch 74 Batch   4/13   train-loss = 1.727
Epoch 74 Batch   10/13   train-loss = 1.715
Epoch 75 Batch   3/13   train-loss = 1.787
Epoch 75 Batch   9/13   train-loss = 1.684
Epoch 76 Batch   2/13   train-loss = 1.762
Epoch 76 Batch   8/13   train-loss = 1.707
Epoch 77 Batch   1/13   train-loss = 1.644
Epoch 77 Batch   7/13   train-loss = 1.616
Epoch 78 Batch   0/13   train-loss = 1.651
Epoch 78 Batch   6/13   train-loss = 1.663
Epoch 78 Batch   12/13   train-loss = 1.670
Epoch 79 Batch   5/13   train-loss = 1.682
Epoch 79 Batch   11/13   train-loss = 1.650
Epoch 80 Batch   4/13   train-loss = 1.655
Epoch 80 Batch   10/13   train-loss = 1.625
Epoch 81 Batch   3/13   train-loss = 1.677
```

```
Epoch 81 Batch    9/13   train-loss = 1.605
Epoch 82 Batch    2/13   train-loss = 1.678
Epoch 82 Batch    8/13   train-loss = 1.596
Epoch 83 Batch    1/13   train-loss = 1.556
Epoch 83 Batch    7/13   train-loss = 1.549
Epoch 84 Batch    0/13   train-loss = 1.582
Epoch 84 Batch    6/13   train-loss = 1.556
Epoch 84 Batch    12/13   train-loss = 1.556
Epoch 85 Batch    5/13   train-loss = 1.589
Epoch 85 Batch    11/13   train-loss = 1.547
Epoch 86 Batch    4/13   train-loss = 1.566
Epoch 86 Batch    10/13   train-loss = 1.531
Epoch 87 Batch    3/13   train-loss = 1.595
Epoch 87 Batch    9/13   train-loss = 1.488
Epoch 88 Batch    2/13   train-loss = 1.530
Epoch 88 Batch    8/13   train-loss = 1.516
Epoch 89 Batch    1/13   train-loss = 1.469
Epoch 89 Batch    7/13   train-loss = 1.413
Epoch 90 Batch    0/13   train-loss = 1.451
Epoch 90 Batch    6/13   train-loss = 1.453
Epoch 90 Batch    12/13   train-loss = 1.449
Epoch 91 Batch    5/13   train-loss = 1.506
Epoch 91 Batch    11/13   train-loss = 1.466
Epoch 92 Batch    4/13   train-loss = 1.487
Epoch 92 Batch    10/13   train-loss = 1.471
Epoch 93 Batch    3/13   train-loss = 1.498
Epoch 93 Batch    9/13   train-loss = 1.420
Epoch 94 Batch    2/13   train-loss = 1.519
Epoch 94 Batch    8/13   train-loss = 1.460
Epoch 95 Batch    1/13   train-loss = 1.462
Epoch 95 Batch    7/13   train-loss = 1.441
Epoch 96 Batch    0/13   train-loss = 1.514
Epoch 96 Batch    6/13   train-loss = 1.456
Epoch 96 Batch    12/13   train-loss = 1.459
Epoch 97 Batch    5/13   train-loss = 1.546
Epoch 97 Batch    11/13   train-loss = 1.507
Epoch 98 Batch    4/13   train-loss = 1.501
Epoch 98 Batch    10/13   train-loss = 1.491
Epoch 99 Batch    3/13   train-loss = 1.552
Epoch 99 Batch    9/13   train-loss = 1.448
Epoch 100 Batch    2/13   train-loss = 1.482
Epoch 100 Batch    8/13   train-loss = 1.441
Epoch 101 Batch    1/13   train-loss = 1.417
Epoch 101 Batch    7/13   train-loss = 1.351
Epoch 102 Batch    0/13   train-loss = 1.389
Epoch 102 Batch    6/13   train-loss = 1.357
```

```
Epoch 102 Batch   12/13   train-loss = 1.370
Epoch 103 Batch   5/13   train-loss = 1.391
Epoch 103 Batch   11/13   train-loss = 1.365
Epoch 104 Batch   4/13   train-loss = 1.380
Epoch 104 Batch   10/13   train-loss = 1.303
Epoch 105 Batch   3/13   train-loss = 1.380
Epoch 105 Batch   9/13   train-loss = 1.347
Epoch 106 Batch   2/13   train-loss = 1.408
Epoch 106 Batch   8/13   train-loss = 1.341
Epoch 107 Batch   1/13   train-loss = 1.306
Epoch 107 Batch   7/13   train-loss = 1.243
Epoch 108 Batch   0/13   train-loss = 1.288
Epoch 108 Batch   6/13   train-loss = 1.305
Epoch 108 Batch   12/13   train-loss = 1.317
Epoch 109 Batch   5/13   train-loss = 1.367
Epoch 109 Batch   11/13   train-loss = 1.326
Epoch 110 Batch   4/13   train-loss = 1.341
Epoch 110 Batch   10/13   train-loss = 1.284
Epoch 111 Batch   3/13   train-loss = 1.360
Epoch 111 Batch   9/13   train-loss = 1.328
Epoch 112 Batch   2/13   train-loss = 1.396
Epoch 112 Batch   8/13   train-loss = 1.357
Epoch 113 Batch   1/13   train-loss = 1.320
Epoch 113 Batch   7/13   train-loss = 1.250
Epoch 114 Batch   0/13   train-loss = 1.240
Epoch 114 Batch   6/13   train-loss = 1.290
Epoch 114 Batch   12/13   train-loss = 1.298
Epoch 115 Batch   5/13   train-loss = 1.324
Epoch 115 Batch   11/13   train-loss = 1.264
Epoch 116 Batch   4/13   train-loss = 1.273
Epoch 116 Batch   10/13   train-loss = 1.220
Epoch 117 Batch   3/13   train-loss = 1.312
Epoch 117 Batch   9/13   train-loss = 1.245
Epoch 118 Batch   2/13   train-loss = 1.277
Epoch 118 Batch   8/13   train-loss = 1.247
Epoch 119 Batch   1/13   train-loss = 1.203
Epoch 119 Batch   7/13   train-loss = 1.157
Epoch 120 Batch   0/13   train-loss = 1.166
Epoch 120 Batch   6/13   train-loss = 1.180
Epoch 120 Batch   12/13   train-loss = 1.172
Epoch 121 Batch   5/13   train-loss = 1.195
Epoch 121 Batch   11/13   train-loss = 1.172
Epoch 122 Batch   4/13   train-loss = 1.193
Epoch 122 Batch   10/13   train-loss = 1.162
Epoch 123 Batch   3/13   train-loss = 1.218
Epoch 123 Batch   9/13   train-loss = 1.151
```

```
Epoch 124 Batch   2/13   train-loss = 1.201
Epoch 124 Batch   8/13   train-loss = 1.185
Epoch 125 Batch   1/13   train-loss = 1.155
Epoch 125 Batch   7/13   train-loss = 1.116
Epoch 126 Batch   0/13   train-loss = 1.123
Epoch 126 Batch   6/13   train-loss = 1.143
Epoch 126 Batch   12/13   train-loss = 1.137
Epoch 127 Batch   5/13   train-loss = 1.178
Epoch 127 Batch   11/13   train-loss = 1.129
Epoch 128 Batch   4/13   train-loss = 1.152
Epoch 128 Batch   10/13   train-loss = 1.120
Epoch 129 Batch   3/13   train-loss = 1.189
Epoch 129 Batch   9/13   train-loss = 1.155
Epoch 130 Batch   2/13   train-loss = 1.162
Epoch 130 Batch   8/13   train-loss = 1.150
Epoch 131 Batch   1/13   train-loss = 1.113
Epoch 131 Batch   7/13   train-loss = 1.076
Epoch 132 Batch   0/13   train-loss = 1.107
Epoch 132 Batch   6/13   train-loss = 1.137
Epoch 132 Batch   12/13   train-loss = 1.095
Epoch 133 Batch   5/13   train-loss = 1.135
Epoch 133 Batch   11/13   train-loss = 1.098
Epoch 134 Batch   4/13   train-loss = 1.146
Epoch 134 Batch   10/13   train-loss = 1.095
Epoch 135 Batch   3/13   train-loss = 1.136
Epoch 135 Batch   9/13   train-loss = 1.106
Epoch 136 Batch   2/13   train-loss = 1.157
Epoch 136 Batch   8/13   train-loss = 1.114
Epoch 137 Batch   1/13   train-loss = 1.088
Epoch 137 Batch   7/13   train-loss = 1.036
Epoch 138 Batch   0/13   train-loss = 1.076
Epoch 138 Batch   6/13   train-loss = 1.086
Epoch 138 Batch   12/13   train-loss = 1.066
Epoch 139 Batch   5/13   train-loss = 1.130
Epoch 139 Batch   11/13   train-loss = 1.089
Epoch 140 Batch   4/13   train-loss = 1.080
Epoch 140 Batch   10/13   train-loss = 1.051
Epoch 141 Batch   3/13   train-loss = 1.132
Epoch 141 Batch   9/13   train-loss = 1.063
Epoch 142 Batch   2/13   train-loss = 1.104
Epoch 142 Batch   8/13   train-loss = 1.092
Epoch 143 Batch   1/13   train-loss = 1.074
Epoch 143 Batch   7/13   train-loss = 1.026
Epoch 144 Batch   0/13   train-loss = 1.077
Epoch 144 Batch   6/13   train-loss = 1.067
Epoch 144 Batch   12/13   train-loss = 1.084
```

```
Epoch 145 Batch    5/13   train-loss = 1.115
Epoch 145 Batch    11/13   train-loss = 1.093
Epoch 146 Batch    4/13   train-loss = 1.088
Epoch 146 Batch    10/13   train-loss = 1.057
Epoch 147 Batch    3/13   train-loss = 1.126
Epoch 147 Batch    9/13   train-loss = 1.053
Epoch 148 Batch    2/13   train-loss = 1.117
Epoch 148 Batch    8/13   train-loss = 1.072
Epoch 149 Batch    1/13   train-loss = 1.049
Epoch 149 Batch    7/13   train-loss = 1.028
Epoch 150 Batch    0/13   train-loss = 1.068
Epoch 150 Batch    6/13   train-loss = 1.054
Epoch 150 Batch    12/13   train-loss = 1.035
Epoch 151 Batch    5/13   train-loss = 1.093
Epoch 151 Batch    11/13   train-loss = 1.054
Epoch 152 Batch    4/13   train-loss = 1.089
Epoch 152 Batch    10/13   train-loss = 1.034
Epoch 153 Batch    3/13   train-loss = 1.093
Epoch 153 Batch    9/13   train-loss = 1.060
Epoch 154 Batch    2/13   train-loss = 1.094
Epoch 154 Batch    8/13   train-loss = 1.064
Epoch 155 Batch    1/13   train-loss = 1.037
Epoch 155 Batch    7/13   train-loss = 1.016
Epoch 156 Batch    0/13   train-loss = 1.016
Epoch 156 Batch    6/13   train-loss = 1.040
Epoch 156 Batch    12/13   train-loss = 1.016
Epoch 157 Batch    5/13   train-loss = 1.040
Epoch 157 Batch    11/13   train-loss = 1.026
Epoch 158 Batch    4/13   train-loss = 1.058
Epoch 158 Batch    10/13   train-loss = 1.014
Epoch 159 Batch    3/13   train-loss = 1.089
Epoch 159 Batch    9/13   train-loss = 1.013
Epoch 160 Batch    2/13   train-loss = 1.026
Epoch 160 Batch    8/13   train-loss = 1.026
Epoch 161 Batch    1/13   train-loss = 1.012
Epoch 161 Batch    7/13   train-loss = 1.008
Epoch 162 Batch    0/13   train-loss = 1.067
Epoch 162 Batch    6/13   train-loss = 1.067
Epoch 162 Batch    12/13   train-loss = 1.083
Epoch 163 Batch    5/13   train-loss = 1.097
Epoch 163 Batch    11/13   train-loss = 0.989
Epoch 164 Batch    4/13   train-loss = 1.056
Epoch 164 Batch    10/13   train-loss = 1.022
Epoch 165 Batch    3/13   train-loss = 1.131
Epoch 165 Batch    9/13   train-loss = 1.068
Epoch 166 Batch    2/13   train-loss = 1.057
```

```
Epoch 166 Batch    8/13   train-loss = 1.041
Epoch 167 Batch    1/13   train-loss = 1.023
Epoch 167 Batch    7/13   train-loss = 1.028
Epoch 168 Batch    0/13   train-loss = 1.053
Epoch 168 Batch    6/13   train-loss = 1.019
Epoch 168 Batch    12/13   train-loss = 1.005
Epoch 169 Batch    5/13   train-loss = 1.054
Epoch 169 Batch    11/13   train-loss = 1.020
Epoch 170 Batch    4/13   train-loss = 1.028
Epoch 170 Batch    10/13   train-loss = 1.003
Epoch 171 Batch    3/13   train-loss = 1.069
Epoch 171 Batch    9/13   train-loss = 1.002
Epoch 172 Batch    2/13   train-loss = 1.068
Epoch 172 Batch    8/13   train-loss = 1.031
Epoch 173 Batch    1/13   train-loss = 0.998
Epoch 173 Batch    7/13   train-loss = 1.000
Epoch 174 Batch    0/13   train-loss = 0.993
Epoch 174 Batch    6/13   train-loss = 0.982
Epoch 174 Batch    12/13   train-loss = 0.986
Epoch 175 Batch    5/13   train-loss = 1.015
Epoch 175 Batch    11/13   train-loss = 1.020
Epoch 176 Batch    4/13   train-loss = 1.045
Epoch 176 Batch    10/13   train-loss = 1.063
Epoch 177 Batch    3/13   train-loss = 1.085
Epoch 177 Batch    9/13   train-loss = 1.003
Epoch 178 Batch    2/13   train-loss = 1.018
Epoch 178 Batch    8/13   train-loss = 0.971
Epoch 179 Batch    1/13   train-loss = 1.015
Epoch 179 Batch    7/13   train-loss = 1.042
Epoch 180 Batch    0/13   train-loss = 1.059
Epoch 180 Batch    6/13   train-loss = 1.031
Epoch 180 Batch    12/13   train-loss = 0.990
Epoch 181 Batch    5/13   train-loss = 1.039
Epoch 181 Batch    11/13   train-loss = 1.032
Epoch 182 Batch    4/13   train-loss = 1.049
Epoch 182 Batch    10/13   train-loss = 1.037
Epoch 183 Batch    3/13   train-loss = 1.051
Epoch 183 Batch    9/13   train-loss = 0.970
Epoch 184 Batch    2/13   train-loss = 0.988
Epoch 184 Batch    8/13   train-loss = 0.988
Epoch 185 Batch    1/13   train-loss = 0.956
Epoch 185 Batch    7/13   train-loss = 0.943
Epoch 186 Batch    0/13   train-loss = 0.943
Epoch 186 Batch    6/13   train-loss = 0.954
Epoch 186 Batch    12/13   train-loss = 0.951
Epoch 187 Batch    5/13   train-loss = 0.965
```

```
Epoch 187 Batch   11/13   train-loss = 0.958
Epoch 188 Batch   4/13   train-loss = 0.949
Epoch 188 Batch   10/13   train-loss = 0.935
Epoch 189 Batch   3/13   train-loss = 0.977
Epoch 189 Batch   9/13   train-loss = 0.921
Epoch 190 Batch   2/13   train-loss = 0.940
Epoch 190 Batch   8/13   train-loss = 0.924
Epoch 191 Batch   1/13   train-loss = 0.920
Epoch 191 Batch   7/13   train-loss = 0.896
Epoch 192 Batch   0/13   train-loss = 0.893
Epoch 192 Batch   6/13   train-loss = 0.912
Epoch 192 Batch   12/13   train-loss = 0.915
Epoch 193 Batch   5/13   train-loss = 0.933
Epoch 193 Batch   11/13   train-loss = 0.897
Epoch 194 Batch   4/13   train-loss = 0.931
Epoch 194 Batch   10/13   train-loss = 0.885
Epoch 195 Batch   3/13   train-loss = 0.939
Epoch 195 Batch   9/13   train-loss = 0.906
Epoch 196 Batch   2/13   train-loss = 0.926
Epoch 196 Batch   8/13   train-loss = 0.900
Epoch 197 Batch   1/13   train-loss = 0.901
Epoch 197 Batch   7/13   train-loss = 0.879
Epoch 198 Batch   0/13   train-loss = 0.883
Epoch 198 Batch   6/13   train-loss = 0.892
Epoch 198 Batch   12/13   train-loss = 0.874
Epoch 199 Batch   5/13   train-loss = 0.903
Epoch 199 Batch   11/13   train-loss = 0.867
```

5 Conclusion

After training with text on Word2Vec and both the models the LSTM model was more efficient and gave better results.We tested with a handful of regularization methods and it was observed that batch normalization was only marginally more efficient than dropout.

References

[1] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[2] Pattara Leelaprute et al. "Does Coding in Pythonic Zen Peak Performance? Preliminary Experiments of Nine Pythonic Idioms at Scale". In: *arXiv preprint arXiv:2203.14484* (2022).

[3] *Recurrent Neural Networks*. https://www.ibm.com/cloud/learn/recurrent-neural-networks. 2020.

[4] Sivasurya Santhanam. "Context based text-generation using lstm networks". In: *arXiv preprint arXiv:2005.00048* (2020).

[5] Ralf C. Staudemeyer and Eric Rothstein Morris. "Understanding LSTM - a tutorial into Long Short-Term Memory Recurrent Neural Networks". In: *CoRR* abs/1909.09586 (2019). arXiv: 1909.09586. URL: http://arxiv.org/abs/1909.09586.

[6] P.J. Werbos. "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560. DOI: 10.1109/5.58337.

[7] Aston Zhang et al. "Dive into Deep Learning". In: *arXiv preprint arXiv:2106.11342* (2021).

[8] Yutao Zhu et al. "Scriptwriter: Narrative-guided script generation". In: *arXiv preprint arXiv:2005.10331* (2020).