DLOps Assignment 2: Ritam Sharma

Transliteration and Multivariate time series forecasting B20BB030

# Seq2Seq

## Question Overview

The main goal is to train a seq2seq model which takes as input Romanized string and produces the

The corresponding word in native script.

## Metrics

### Accuracy

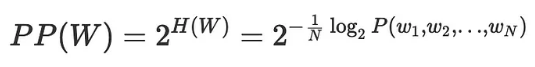
Accuracy is a commonly used metric for evaluating classification models, including those based on LSTM. In the context of LSTM models, accuracy is typically used to measure the percentage of correctly classified instances from the total number of instances.

To calculate accuracy for an LSTM model, I first need to split your dataset into a training set and a test set. You then train the LSTM model on the training set and use it to predict the labels of the instances in the test set. You can then compare the predicted labels with the true labels and count the number of correct predictions.

### Perplexity

Perplexity is often used as a metric for evaluating language models, which are a type of seq2seq model that generate sequences of words or tokens. Perplexity measures how well the language model can predict the next token in a sequence, given the previous tokens. A lower perplexity indicates that the language model is better at predicting the next token and thus has a better understanding of the underlying language.

Perplexity can also be defined as the exponential of the cross-entropy:



The perplexity measures the amount of “randomness” in our model. If the perplexity is 3 (per word) then that means the model had a 1-in-3 chance of guessing (on average) the next word in the text. For this reason, lower value of perplexity means better performance

# Analysis

## Data Exploration

The Dakshina dataset is a collection of text in both Latin and native scripts for 12 South Asian languages. For each language, the dataset includes a large collection of native script Wikipedia text, a romanization lexicon which consists of words in the native script with attested romanizations, and some full sentence parallel data in both a native script of the language and the basic Latin alphabet.

# Methodology

* Data preparation: In order to train and test the seq2seq model, collect a dataset of paired input-output sequences. Data preparation and cleaning, including text normalization and tokenization.
* ASCII values are taken for Hindi letter tokenization and a padding token with the value 0 is added to every sequence
* Split the data into training and validation and testing sets: Separated the dataset into a training set for the model's training and a validation set for fine-tuning hyperparameters and avoiding overfitting.
* Define the encoder and decoder: The input sequence is taken in by the encoder, which converts it into a fixed-length vector that the decoder uses to produce the output sequence. The decoder creates the output sequence one token at a time using the encoder's output.
* Define the architecture: The input sequence is taken in by the encoder, which converts it into a fixed-length vector that the decoder uses to produce the output sequence. The decoder creates the output sequence one token at a time using the encoder's output.
* Train the model: Use a suitable optimisation technique, such as stochastic gradient descent (SGD) or Adam, to train the seq2seq model on the training set. Examine the model's performance on the validation set, and adjust the hyperparameters as necessary.
* Evaluate the model: Determine the model's accuracy, precision, recall, and other performance measures by assessing the model's performance on the test set once it has been trained.
* Refine the model: To enhance the model's performance, make minor adjustments to the architecture, hyperparameters, or other components. The training and evaluation process should be repeated until the desired performance level is reached.

## Loss Function used : Cross Entropy

The basic idea behind cross-entropy is to measure the difference between the predicted probability distribution and the true probability distribution of the labels. The loss function aims to minimize this difference, which is often referred to as the "cross-entropy loss".

The cross-entropy loss is defined as follows:

L = -1/N \* Σ(y \* log(y\_hat) + (1-y) \* log(1-y\_hat))

where:

* L is the cross-entropy loss
* N is the number of samples in the dataset
* y is the true label (0 or 1)
* y\_hat is the predicted probability of the label being 1

The cross-entropy loss penalizes the model more heavily when it makes confident incorrect predictions. For example, if the true label is 0 and the model predicts a probability of 0.9 for label 1, the loss will be larger than if the model predicts a probability of 0.6 for label 1.

# Results

## LSTM 1

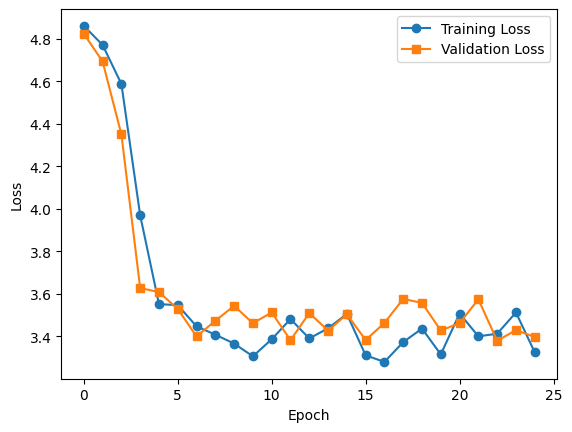
Input embedding size: 16

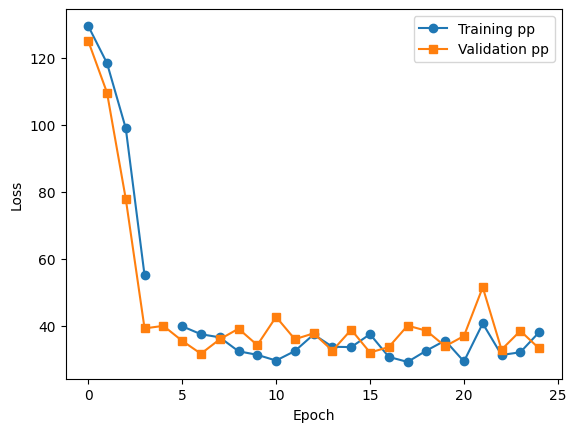
number of encoder layers: 1

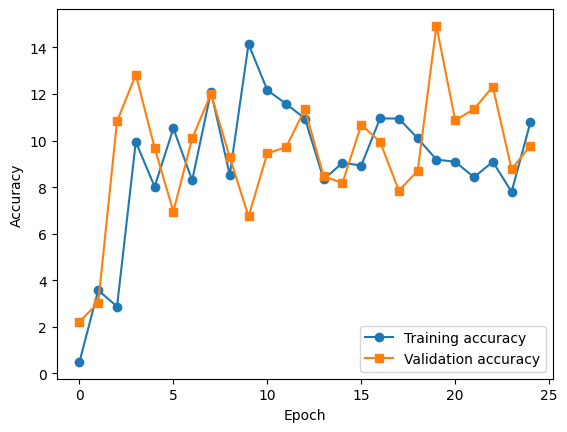
number of decoder layers: 1

hidden layer size: 16

Dropout=1(no dropout)







Perplexity: 28.865510046482086

Accuracy: 10.98901098901098

## LSTM2

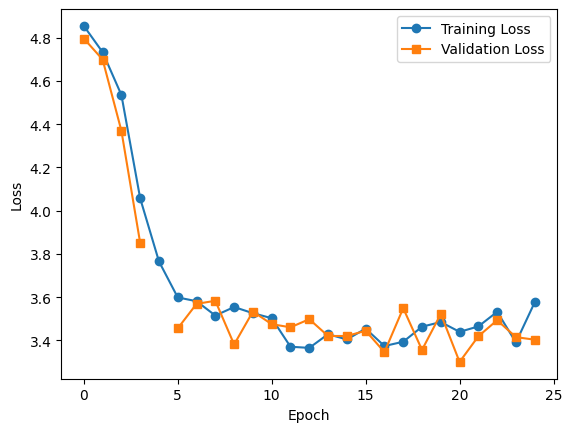
Input embedding size: 64

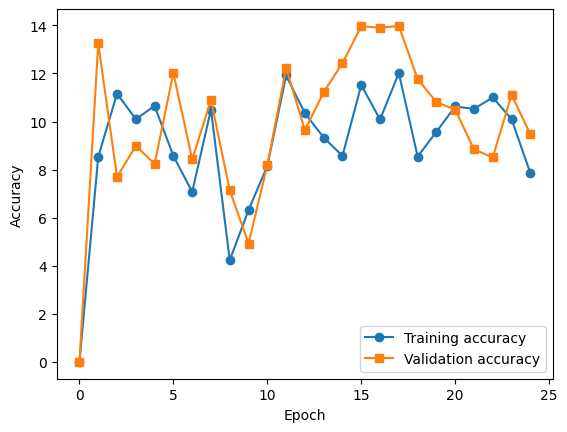
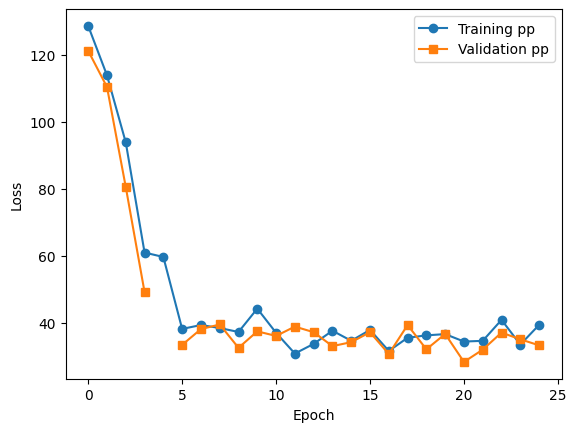
number of encoder layers: 3

number of decoder layers: 3

hidden layer size: 64

Dropout=1(no dropout)





Perplexity: 28.224501967430115

Accuracy: 13.089005235602095

## LSTM 3

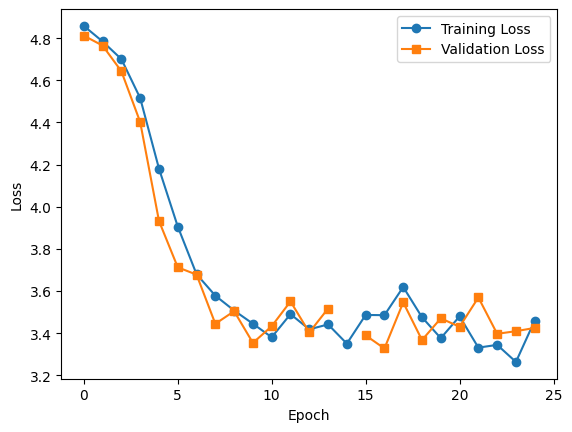
Input embedding size: 16

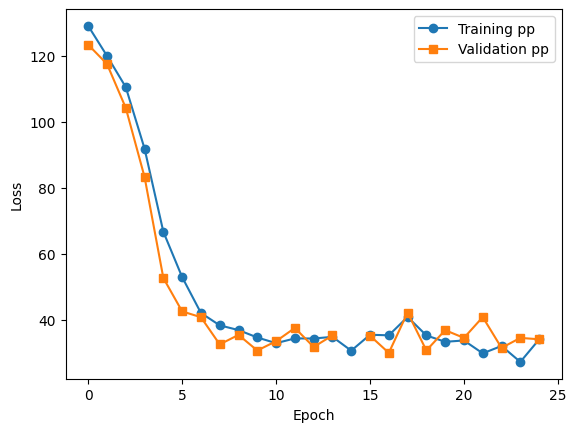
number of encoder layers: 1

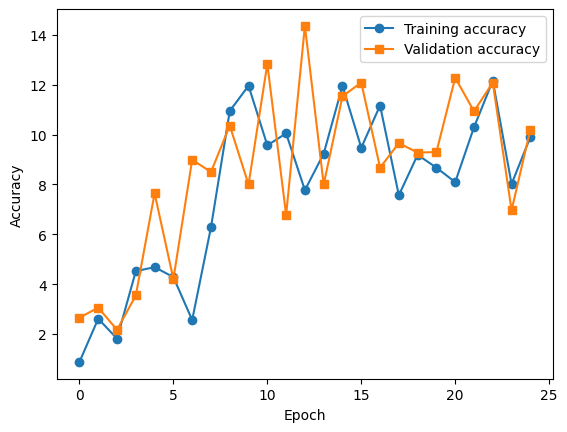
number of decoder layers: 1

hidden layer size: 16

dropout : 0.1







Perplexity: 23.384103655815125

Accuracy: 14.285714285714286

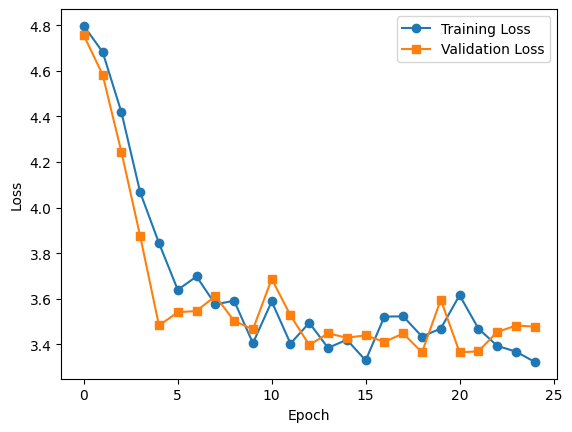
## RNN1

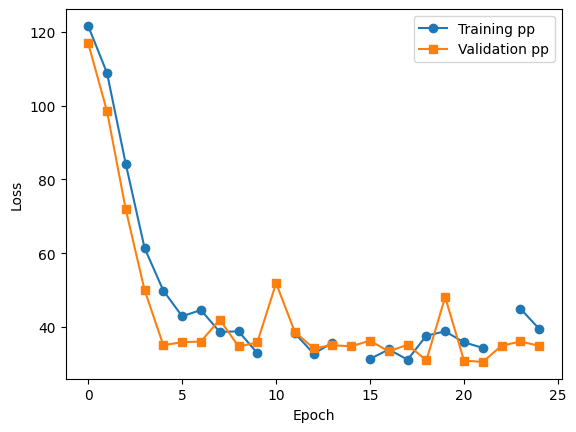
Input embedding size: 16

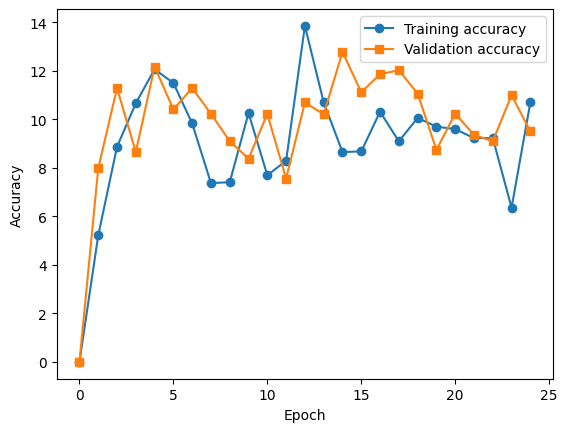
number of encoder layers: 1

number of decoder layers: 1

hidden layer size: 16







Perplexity: 47.52447843551636

Accuracy: 9.090909090909092

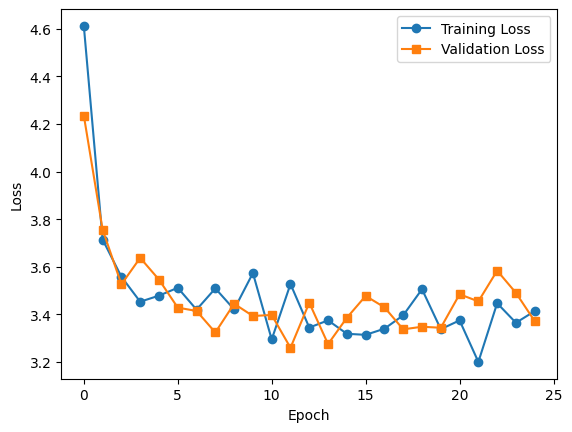
## RNN2

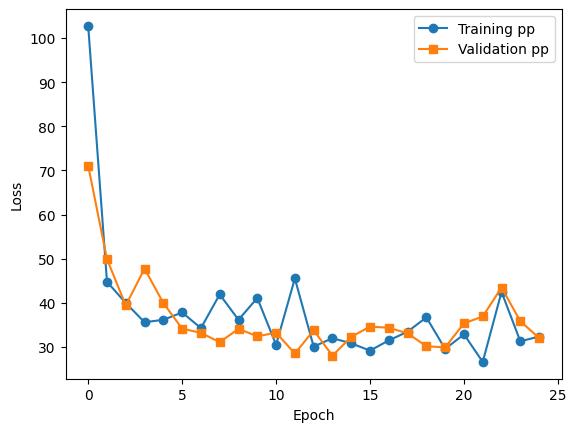
Input embedding size: 64

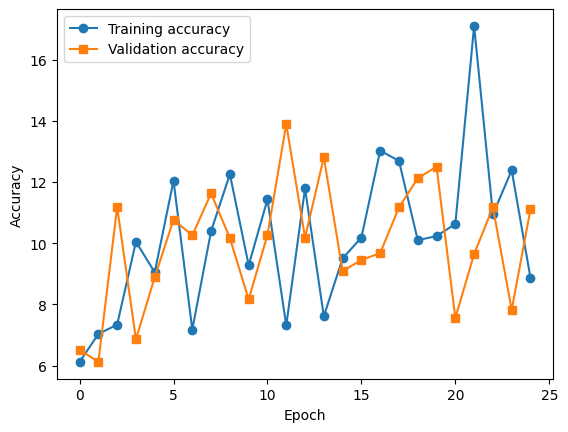
number of encoder layers: 3

number of decoder layers: 3

hidden layer size: 64







Perplexity: 36.249361366033554

Accuracy: 10.112359550561798

## RNN3

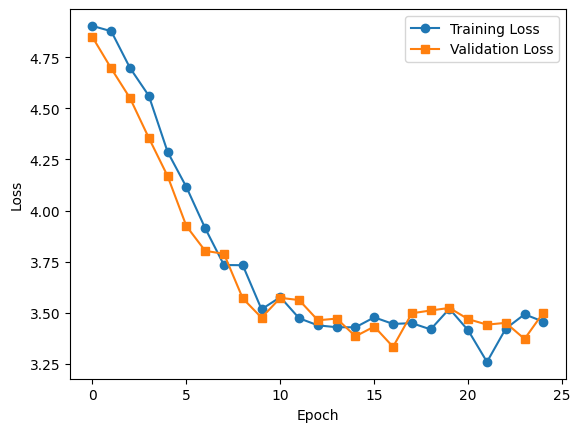
Input embedding size: 16

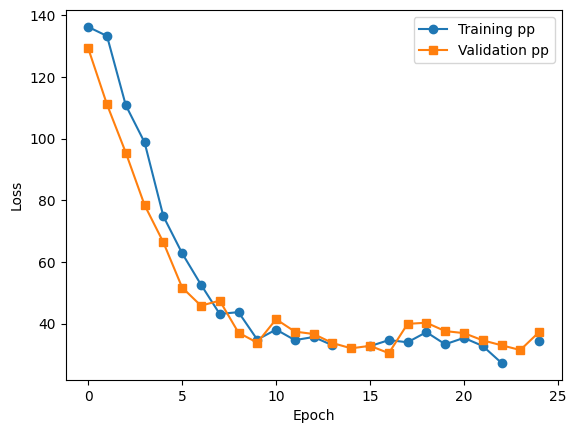
number of encoder layers: 1

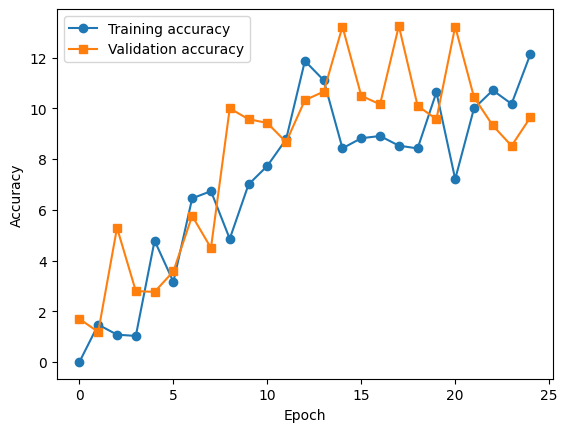
number of decoder layers: 1

hidden layer size: 16

dropout=0.1(on both)







Perplexity: 38.86574399471283

Accuracy: 11.11111111111111

# Results Comparison and Discussion

## LSTM versus RNN Convergence

|  |  |  |
| --- | --- | --- |
| LSTM | RNN | Description/Hyperparameter |
|  |  | Loss curves : Input embedding size: 16  number of encoder layers: 1  number of decoder layers: 1  hidden layer size: 16 |
|  |  | Input embedding size: 64  number of encoder layers: 3  number of decoder layers: 3  hidden layer size: 64 |
|  |  | Input embedding size: 16  number of encoder layers: 1  number of decoder layers: 1  hidden layer size: 16  dropout : 0.1 |

Discussion

* Overall, LSTM seems to converge faster than LSTM besides the instance where the embedding size is 64, Input embedding size is 64, the number of encoder layers is 3, the number of decoder layers is 3, hidden layer size is 64.   
    
  We can observe that LSTM has a steeper learning curve slope, therefore, we can say that it converges faster than LSTM. This can be attributed to problems such as the vanishing gradient problem.
* LSTM networks achieve this by using gates to control the flow of information through the network, allowing it to selectively remember or forget previous inputs. This helps to prevent the gradients from vanishing or exploding, which can slow down or even prevent convergence.
* However, as we observed this isn’t always true. it's important to note that the convergence rate also depends on the specific architecture, hyperparameters, and training data, so it's not always the case that LSTMs will converge faster than RNNs in every situation.
* Furthermore I observed that RNN take less time per epoch even though they converge at a higher epoch

## Dropout and Model performance

|  |  |  |
| --- | --- | --- |
| Without Dropout | With Dropout (=0.1) | Description/hyperparameters |
| Perplexity: 28.865510046482086  Accuracy: 10.989010989010989 | Perplexity: 23.384103655815125  Accuracy: 14.285714285714286 | LSTM model Input embedding size: 16  number of encoder layers: 1  number of decoder layers: 1  hidden layer size: 16 |
| Perplexity: 47.52447843551636  Accuracy: 9.090909090909092 | Perplexity: 38.86574399471283  Accuracy: 11.11111111111111 | RNN model Input embedding size: 16  number of encoder layers: 1  number of decoder layers: 1  hidden layer size: 16 |

Discussion

* Overall, LSTM performs much better than RNN in terms of perplexity and accuracy.
* And from the proof above, It can be concluded that dropout increases performance in both RNN and LSTM
* Dropout can be a powerful regularization technique for improving the performance of LSTM and RNN models by reducing overfitting and improving generalization ability, resulting in better accuracy and effectiveness for a variety of tasks.

## Performance and Hidden dimension

|  |  |  |
| --- | --- | --- |
| Hidden layer size: 16 | Hidden layer size 64 | Description |
| Perplexity: 28.865510046482086  Accuracy: 10.989010989010989 | Perplexity: 28.224501967430115  Accuracy: 13.089005235602095 | LSTM model |
| Perplexity: 47.52447843551636  Accuracy: 9.090909090909092 | Perplexity: 36.249361366033554  Accuracy: 10.112359550561798 | RNN model |

Discussion

* Overall, LSTM performs much better than RNN in terms of perplexity and accuracy.
* And from the proof above, It can be concluded that smaller hidden layer size performs worse has compared to a bigger hidden layer size
* Increasing the hidden layer size in LSTM and RNN models can lead to better performance because it allows the models to learn more complex representations of the input data. LSTM and RNN models are designed to handle sequential data with temporal dependencies, and the hidden layer is where the model learns to encode the input sequence information. A larger hidden layer size means that the model can learn more intricate and detailed representations of the input data, which can lead to better performance on tasks that require more complex patterns.
* By increasing the hidden layer size, the model can capture more nuances and relationships in the input data, which can improve the model's ability to make accurate predictions or generate output sequences. This is particularly beneficial for tasks that require long-term dependencies, where the model needs to remember information from earlier in the input sequence to make accurate predictions.
* However, it's worth noting that increasing the hidden layer size also increases the model's complexity, which can make training more difficult and require more computational resources. Additionally, increasing the hidden layer size beyond a certain point can lead to diminishing returns or even overfitting, where the model begins to memorize the training data rather than learning general patterns.

## Attention network added to LSTM

|  |  |
| --- | --- |
| Perplexity loss over epoch | Loss curve and Accuracy curve |
|  |  |
|  |  |

|  |  |
| --- | --- |
| Metrics | Values |
| Perplexity | 21.09592767804861 |
| Accuracy | 15.675675675675675 |

Discussion

Out of all the models, the attention-based LSTM model performs the best with the lowest perplexity and highest accuracy as shown above

Attention-based LSTM models can perform better than traditional LSTM models for a variety of tasks because they allow the model to focus on the most relevant parts of the input sequence when making predictions or generating output sequences.

Traditional LSTM models use a fixed-length hidden state to encode the entire input sequence. This can be problematic for tasks that involve long sequences or where different parts of the sequence are more important than others. Attention-based LSTM models address this issue by incorporating an attention mechanism that dynamically weights the importance of different parts of the input sequence based on the current state of the model.

The attention mechanism allows the model to selectively focus on the most relevant parts of the input sequence for each step of the prediction or generation process. This can improve the model's ability to capture long-term dependencies and make accurate predictions or generate output sequences.

Additionally, attention-based LSTM models can be more interpretable than traditional LSTM models because they explicitly show which parts of the input sequence the model is attending to at each step of the prediction or generation process. This can be useful for tasks where understanding the model's decision-making process is important.

Overall, attention-based LSTM models can perform better than traditional LSTM models because they allow the model to selectively focus on the most relevant parts of the input sequence and capture long-term dependencies more effectively. This can lead to improved accuracy and effectiveness for a variety of tasks.

# Conclusion

From the above, results and discussions we can conclude that LSTM with attention and dropout performs the best

9 Refences used :-

https://towardsdatascience.com/perplexity-in-language-models-87a196019a94#:~:text=Perplexity%20can%20also%20be%20defined,based%20on%20the%20cross%2Dentropy%3F

# Multivariate Series prediction

## Data Exploration

The Individual household electric power consumption Data Set is a time-series dataset that contains measurements of electric power consumption in one household with a one-minute sampling rate. The dataset spans over a period of almost four years, from December 2006 to November 2010, and includes 2,075,259 observations.

The dataset was collected by a single-phase energy meter, measuring the household's electric power consumption with a resolution of one minute. The meter measured the voltage and current in the house's main circuit, and from this, the active power, reactive power, and apparent power were calculated. In addition to the power measurements, the dataset also includes measurements of global active power, voltage, and current intensity.

The dataset includes 9 attributes, which are:

* Date: the date on which the measurement was taken (format dd/mm/yyyy)
* Time: the time at which the measurement was taken (format hh:mm:ss)
* Global\_active\_power: the household's total active power consumption in kilowatts (kW)
* Global\_reactive\_power: the household's total reactive power consumption in kilowatts (kW)
* Voltage: the average voltage (in volts) measured over one minute
* Global\_intensity: the average current intensity (in amps) measured over one minute
* Sub\_metering\_1: the active power consumption (in kilowatts) in the kitchen area
* Sub\_metering\_2: the active power consumption (in kilowatts) in the laundry area
* Sub\_metering\_3: the active power consumption (in kilowatts) in the climate control system

## Data processing

The preprocessing consists of the following steps:

* The 'read\_csv' function is used to read the CSV file, and a number of arguments are supplied to it, including the CSV file's delimiter, the columns that should be parsed as dates, the way to handle missing values, and the data types of pertinent fields.
* The 'dropna()' function is used by the code to eliminate any rows with missing values after reading in the CSV file.
* The code then uses Pandas' to numeric function to change the data types of many columns to "float." To make sure the data is in the right format for analysis, this is done. Date and time features are merged and used as index
* A window size of 5 is chosen, such that each training example contains a batch of 5 datapoints that have 7 features which will be used to predict the next datapoint. Along with batch of 5, the training example will also contain the true label which the model tries to predict

## Implementation of Multivariate series prediction

* Data Preparation: Gather the dataset and preprocess itas mentioned above. Sets for training, and testing are made using the dataset class to make sure that all variables have the same scale, it is also crucial to normalize the data.
* Define the LSTM Model: Build a model architecture that processes the data using LSTM layers. The number of time steps and features in the input data should be considered by the model. In order to improve the performance of the model, you can experiment with various hyperparameters.
* Train the LSTM Model: Use the training set to train the LSTM model. You can employ a variety of optimisation techniques, such as Adam or SGD, and a suitable loss function, such as mean square error (MSE).
* Validate the LSTM Model: Use the validation set to evaluate the model's performance. To evaluate the model's correctness, you can keep an eye on various performance indicators including mean absolute error (MAE), root mean square error (RMSE), or coefficient of determination (R2).
* Test the LSTM Model: Use the testing set to evaluate the model's effectiveness. Use the same performance measures from the validation step to evaluate the model.
* Tune the LSTM Model: I altered the architecture or hyperparameters based on the model's performance to increase the model's accuracy.

## Results and discussion

|  |  |  |  |
| --- | --- | --- | --- |
| Loss Curves | Real global activepower and predicted global active power for the testing days | MAE values | DatasetSplit |
|  |  | 2.47404797872 | 80:20 |
|  |  | 4.15742511107 | 70:30 |

**Discussion**

We can observe that we MAE value Is quite less for the model with 80:20 split as compared to the model with 70:30 split

Furthermore, the predicted and actual plot for 70:30 is a lot more shifted above as compared to 80:20

# Steps to Run it on IITJ server in a container

1. Change the password

2. create a directory on local containing py file, dataset, slurm file

3. connect with ssh

4. copy data from local to host using scp command

5. submit batch job

6. do output.log()

# 

# The above is the output.log() of the py file