**Introduction For thesis?**

Cryptocurrency is a digital asset that does not need intermediatory for transactions, unlike fiat currency. The data is encrypted and stored in the blockchain–distributed ledger connected with a network of computers. The advantage of these digital assets is that the transfer is faster and cheaper, but these are highly volatile. First-ever cryptocurrency presented was Bitcoin by Satoshi Nakamoto in November 2008 through his white paper called “Bitcoin: peer-to-peer Electronic Cash System” [paper link] . It was followed by other/alternative cryptocurrencies (altcoins) such as Ethereum (ETH), Solana (SOL), etc. These assets have gained a lot of attraction from investors in the last decade. As of this writing, BTC stands at $29,951 with a market cap of $570B, the highest market share in the crypto ecosystem followed by ETH. Analyzing the price fluctuation and time series has always been a topic of great interest in the econometrics and machine learning community. It gives the direction to the potential trading strategies for maximum profits to individuals/firms. The nature of crypto time series with high fluctuation and volatility requires sophisticated modeling.

The last decade has seen an unparallel advancement in natural language processing (NLP) and deep learning. The first significant discovery came in 2013 with the vector representation of words called embeddings. It gave the models a performance boost. In the next couple of years, recurrent neural networks (RNN) and long short-term memory (LSTM) gained popularity because of their ability to process sequential text data. LSTMs were better than RNN since they partially resolved the 'vanishing gradient' problem and were able to process larger historical data. Then in 2017, a specific attention-based network was introduced called Transformers that replaced RNN and LSTMS as the state-of-the-art models for sequence modeling and machine translation and has been superiorly dominant in language/sequential modeling. Sequential modeling is the ability or task to interpret, model, or predict the next data point based on historical points. Time series have data points in a constant interval of time that categorizes them in the category of sequential data. The architectures used in the sequence modeling in NLP are modified and used by the researchers in the time series forecasting. These models have shown excellent results in this field, with LSTMs being the go-to model. With the advent of Transformer, the question arises: is Transformer better than LSTM in time series?

Therefore, the focus of this thesis would be on forecasting the hourly bitcoin prices in USD with Transformers and comparing the results with LSTMS. I chose BTC since the entire crypto market is correlated with the BTC price fluctuation and is the most valuable crypto currency currently.

(Improve introduction)

**What is Transformer architecture?**

Transformer was first introduced in the paper Attention Is All You Need by xyz. It is a novel architecture which was able to handle long range dependencies in **sequence-to-sequence** modelling solely based on attention mechanism without using convolution or recurrence. This helped the architecture be more parallelizable and taking less time to train compared to the state of the art of models such as recurrent neural network, long short-term memory, and gated recurrent neural networks. RNN and LSTM were the state-of-the-art architecture in machine translation and language modelling. The basic working of these architectures was sequential, creating hidden states from the current input and the last hidden state. This sequential processing did not allow much room for parallel processing, which was crucial for handling long range dependencies. Transformer was able to achieve state of the art result on WMT 2014 English-to-German translation task and WMT 2014 English-to-French translation task.

The Transformer architecture consists of encoder and decoder block. Each of these blocks consist of multiple identical units of encoder and decoder stack on top of each other. The number of units in both blocks are same which is a hyper parameter and was chose as 6 in the original paper.

Diagram

Description automatically generated

**Encoder and Decoder?**

Fig 1 from [] shows one unit of encoder on the left half and one unit of decoder on the right half. Each encoder consists of two sub-units, multi-head self-attention and feed forward network. Feed forward network consist of two transformations with ReLu activation in between. After every subunit there is residual connection followed with a layer normalization. The decoder also consists of the same sub-units with an additional sub-unit that performs a masked multi-head attention to not allow it from attending to the words/data-points later in the series and prevent the information leak. The multi-head self-attention mechanism is also modified because it performs operation over the output from encoder. All the sub-unit, including the embedding, produces output of dimension dmodel to incorporate the residual connection.

**What is Attention?**

Attention is a mechanism which try to depict how the human brain focuses on important thing. To understand this one standard example is the following sentence: **“The animal didn’t cross the street because it was too tired”**. For humans it is easy to understand the word “it” refers to the animal but how do you make the machines understand this? This is where the attention plays part.

To understand how the calculation in attention works let’s take an example of shorter sentence: **“Come here”**. When processing the first word the steps are as follows:

1. Calculate the Query, Key, and Value vector for each of these words embedding.
2. Calculate the dot product of query vector and the key vector
3. Divide the score by the square root of the dimension of the key (dk) which in the paper was 64, hence 8 here.
4. Calculate the SoftMax of the result from previous steps
5. Multiply the SoftMax score with the value vector
6. Point wise summation of value vector.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Word** | **Query** | **Key** | **Value** | **Score** | **Score / 8** | **Softmax** | **Softmax\*V** | **Sum** |
|  | **Step 1** | | | **Step 2** | **Step 3** | **Step 4** | **Step 5** | **Step 6** |
| Come | q1 | k1 | v1 | q1.k1 | q1.k1/8 | s1 | s1\*v1 | z1 |
| Here |  | k2 | v2 | q1.k2 | q1.k2/8 | s2 | s2\*v2 |  |

Similar steps are done for the next two words and the attention vector is calculated. These vectors z1, z2, z3, are then passed to the feed forward neural network. In the actual implementation this is done in matrices for better performance.

![Text

Description automatically generated]()

How is Query, Key and Value matrices calculated? It is done by first stacking our word embeddings into a matrix form and then multiplying it with the trainable weight matrices WQ, WK, WV respectively. This attention mechanism is not performed once but h times which in the original paper was 8, hence it is termed as multi headed attention. This would give eight different set of randomly initialized WQ, WK, WV weight matrices projecting word embeddings into different subspaces. Each head produces attention matrix which are then concatenated and passed through another linear projection with matrix W0, since feed forward layer is not Diagram

Description automatically generatedexpecting a concatenated matrix. **(some more details about MH attention)**

**Positional encoding?**

Since all the sequence of word is processed in a chunk, transformers need a sense of the order of the words or relative position in the sequence. To resolve this, it adds a position vector to each input embedding at the bottom of the encoder and decoder stacks. They have the same dimension as the input embeddings so that they can be summed up. In the original paper the authors used sine and cosine function of different frequencies as given below:

![Text

Description automatically generated]()

Where pos is the position of the word and I is the dimension. To understand this let’s look at an example from [[3]](#positionencoding). The sentence is ‘I am a robot’ with dmodel = 4 and instead of 10000 it uses 100.

![Table

Description automatically generated]()

These positional encoding are added to the word embeddings and passed to the network.

(Add paragraph about masking)

**Related work:**

[Li et al., 2019] focuses on improving the vanilla architecture for forecasting. It targets the self-attention dot product by introducing convolution to generate queries and keys and capture the local context. Further, it addresses the memory usage of default architecture which 0(L2) and introduces the log sparse mask which that reduces the complexity to O(LlogL)

[This paper](#LSTMBITCON): Performs the bitcoin price prediction using ensemble technique on LSTM. They trained three different LSTMS on different time granularity (minutes, hours, and days) and combined the results. The idea is to catch both short term with fine granularity and long-term trend with coarse granularity. The proposed model works better than classical statistical methods, such Auto-Regression, Moving Averages, and Auto Regressive Integrated Moving Averages (ARIMA) and captures the real-world fluctuation.

[This paper](#Dogecoin): works on the price forecasting of the doge coin using Transfomer but modifies the default positional encoding with the Time2vec [paperlink] that includes the periodic and non-periodic representation. Moreover, it uses both encoder and decoder with no attention module in decoder unlike default architecture.

[This paper:](#RNN_AND_LR) Author compares the performance of RNN and linear regressions on minute-by-minute data of bitcoin from 2013 to 2018.

[This thesis](#TS_Transformer_Thesis): presents the problem as a classification problem by binning the price change in an offset in 4 different classes. Uses exponential smoothing for feature generations and volume weighted average prices. Creates a strong LSTM model for baseline and achieves better results with Transformer architecture.

References: (APA)

* <https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/>
* <https://jalammar.github.io/illustrated-transformer/>
* <https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/#:~:text=Transformers%20use%20a%20smart%20positional,summed%20with%20its%20positional%20information>.
* <https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf>
* <https://towardsdatascience.com/high-level-history-of-nlp-models-bc8c8b142ef7>
* [Li et al., 2019] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In NeurIPS, 2019.
* Performance Evaluation of Machine Learning Algorithms for Bitcoin Price Prediction
* M. Shin, D. Mohaisen, and J. Kim, “Bitcoin Price Forecasting via Ensemble-based LSTM Deep Learning Networks,” in 2021 International Conference on Information Networking (ICOIN), pp. 603–608, 2021.
* Multi-Head Self-Attention Transformer for Dogecoin Price Prediction
* <https://www.investopedia.com/terms/c/cryptocurrency.asp>
* Performance Evaluation of Machine Learning Algorithms for Bitcoin Price Prediction
* Predicting time series with Transformer – (Thesis)

**Parts:**

1. **Introduction**
2. **Related work (Vanilla transformer, LSTM)**
   1. **Papers**
   2. **architeture**
3. **Solution Approach (changes to the transformer architecture) 02/3**
4. **Evaluation 2/3**
5. **Future work (last chapter)**
6. **Conclusion**