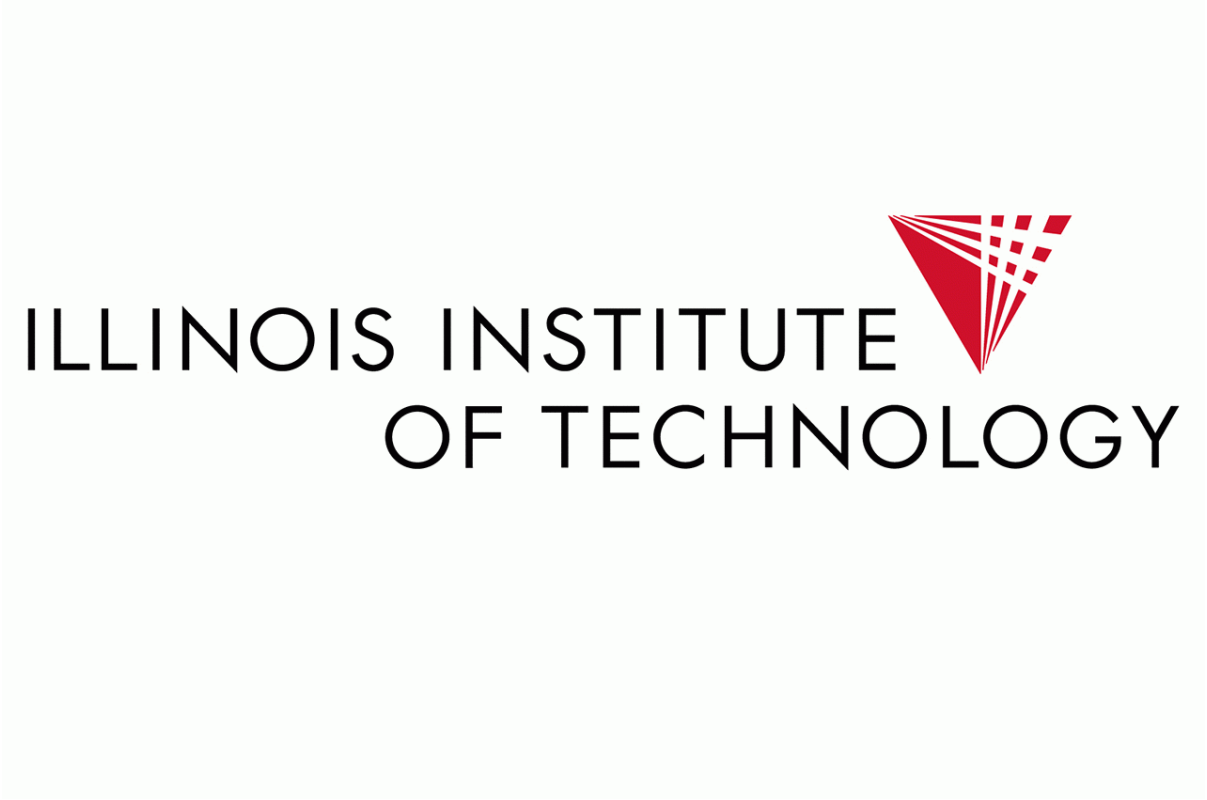
CS 584 - Machine Learning

Instructor: Steve Avsec

Project Members:

1. Ahad Hussain – A20543489
2. Usman Matheen Hameed - A20564338
3. Sreehari Thirumalai Bhuvaraghavan - A20560224

Project: Next Token Prediction using Self Attention Mechanism



1. Problem Statement

To implement and explore the Decoder part of this Transformer model, which is pivotal in generating outputs in sequence-to-sequence tasks. This implementation aims to focus on the next token prediction methodology, whereby the model predicts subsequent elements of the sequence based on prior information and context.

2. Model Architecture

In the customized implementation of the Transformer Decoder, the architecture has been specifically tailored to focus on Masked Multi-Head Attention without incorporating the standard Multi-Head Attention mechanism that typically processes the Encoder's output. This adaptation is particularly relevant for tasks involving auto-regressive prediction, where the model generates one token at a time and must not peek at future tokens.

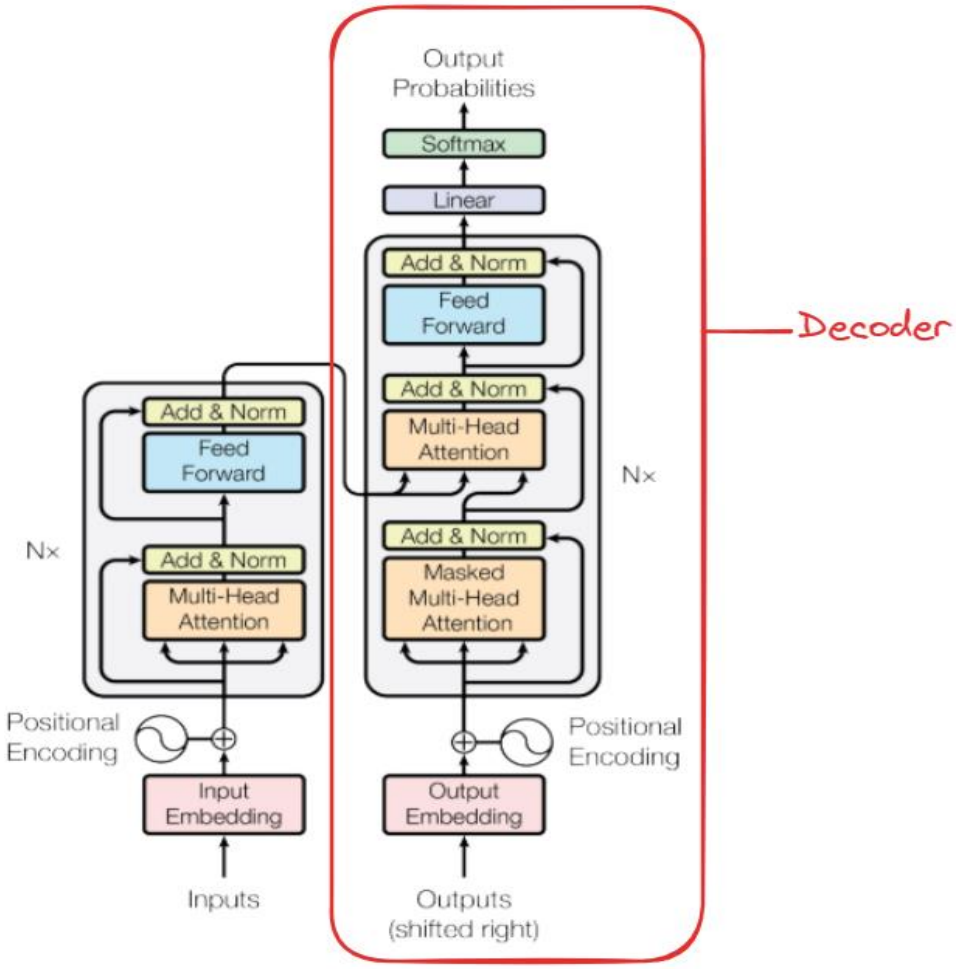


Figure – Transformer Model

2.1. Decoder Architecture

The Decoder configuration consists of several layers structured to optimize the next token prediction:

Number of Decoder Layers: There are 6 layers in the Decoder, maintaining consistency with standard practices that balance computational efficiency and learning capacity.

Number of Attention Heads: Each Decoder layer utilizes 6 attention heads. This setup allows the model to focus on different positions of the input sequence simultaneously, enhancing its ability to capture various linguistic features and dependencies.

Embedding Size: The model employs an embedding size of 384. This dimensionality is crucial for encoding the input tokens into a dense vector space before processing through the attention layers.

2.2. Masked Multi-Head Attention Implementation

The core component of this Decoder is the Masked Multi-Head Attention mechanism. Here, attention weights are adjusted to ensure that during training, the prediction for a particular token can only be influenced by known outputs, i.e., the tokens that precede it. This is crucial for the model to effectively learn sequential dependencies without relying on future context, mimicking the generation of language in a left-to-right manner.

The specific implementation detail provided highlights the application of a triangular mask to the attention weights:

```
weights = weights.masked_fill(self.tril[:T,:T].to('cuda:0') == 0, float("-inf")).to('cuda:0')
```

This code snippet indicates that the attention mechanism employs a lower triangular matrix (self.tril[:T,:T]) to mask future tokens during training, where T represents the sequence length. This matrix is used to set the attention scores of future tokens to negative infinity (float("-inf")), effectively removing them from consideration during the softmax calculation, which precedes the output generation. The operation is performed on the CUDA GPU (cuda:0) to enhance processing efficiency.

3. Data Preparation

For the implementation of the Transformer Decoder focused on next token prediction, the "tiny_shakespeare" dataset was utilized. This dataset is a

compact compilation derived from the works of William Shakespeare, encompassing a rich variety of his plays and poetry. The tiny_shakespeare dataset features approximately 40,000 lines of text and it is specifically tailored for tasks that involve character sequence prediction, making it an ideal choice for training models on the nuances of early modern English text.

For data preparation, initially all unique characters in the text were identified and used to create a vocabulary. Each character was then mapped to a unique integer, facilitating the conversion of textual data into a numerical format that the model could process. The entire corpus was encoded into a sequence of integers. This encoded data was split into training and validation sets, with 90% of the data allocated for training and the remaining 10% for validation. The data loading function was designed to dynamically generate batches for training or validation. Each batch consisted of a segment of the encoded data (specified by the block size) along with the subsequent characters, which serve as the targets for the model to predict. This setup allows the model to learn to predict the next character in a sequence, essential for training on tasks like text generation based on historical data input.

4. Training

The training of the model utilized the tiny_shakespeare dataset to facilitate character sequence prediction. The specific training setup and methodology employed are outlined below:

4.1. Training Parameters

Batch Size: 64, allowing for sufficient sample variability while maintaining manageable memory usage.

Block Size: 256, determining the length of the sequence each training example would cover.

Maximum Iterations: 2,500, defining the total number of training cycles the model undergoes.

Evaluation Interval: Every 500 iterations, which helps monitor the model's performance and generalization ability over time.

Learning Rate: 0.0001, chosen to optimize convergence speed without overshooting minima.

Evaluation Iterations: 200, used during each evaluation phase to assess the model’s performance reliably.

Embedding Size: 384, ensuring a dense representation of character inputs.

Number of Heads: 6, facilitating the model's ability to pay attention to different parts of the input sequence.

Number of Layers: 6, providing depth to the model's learning capability.

Parameter Count: The total trainable parameters were calculated, giving insights into the model's complexity and computational demands.

Optimizer: The optimizer was AdamW, which is the most common optimizer that is used with deep learning models.

5. Evaluation

The primary metric used to assess the model's performance was **cross-entropy loss**, which quantifies the difference between the predicted probabilities and the actual labels. This metric is particularly suitable for classification problems like next token prediction, where the goal is to minimize the prediction error over iterations.

5.1. Data Distribution

The dataset was split into two subsets for training and validation purposes:

Training Data: 90% of the total data, used to train the model.

Validation Data: 10% of the data, used to evaluate the model's performance on unseen data.

5.2. Training and Validation Outcomes

During the first training run with 10 million parameters, the model exhibited instability. This instability could be indicative of several issues, including overfitting, insufficient regularization, or suboptimal hyperparameter settings.

5.3. Adjustments and Challenges

In an effort to stabilize and generalize the model, several adjustments were made.

Increased Number of Attention Heads and Decoder Blocks: These changes were intended to enhance the model's learning capability and its ability to capture more complex patterns in the data.

Modified Learning Rate: The learning rate was adjusted to improve the training dynamics.

Increased Embedding Size: The embedding size was increased from 384 to 512 to allow for richer representations of input tokens.

5.4. Other Evaluation Metric

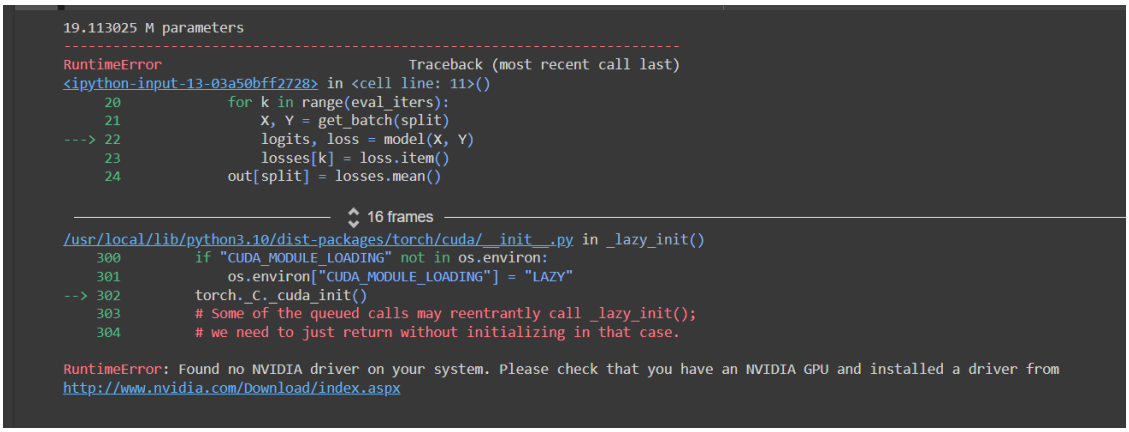
Perplexity: Often used in language modeling, perplexity measures how well a probability distribution or probability model predicts a sample. It is a transformation of the cross-entropy loss, providing a more interpretable measure that indicates the average branching factor of a language model. A lower perplexity score indicates a model that can predict the next token with higher certainty, which is desirable in next token prediction tasks.

6. Limitations

Computational Limits: The initial attempt to run the enhanced model on a laptop with an RTX 4060 GPU was halted due to overheating, which suggests that the computational demand exceeded the hardware's memory capacity.

Resource Exhaustion: A subsequent attempt using Google Colab failed due to 'CUDA out of memory' errors, indicating that the resource requirements of the model with its increased complexity surpassed the available GPU memory.

Model with 19M parameters



7. Results

The initial training perplexity was 62.2568, and validation perplexity was 63.4339. Over the course of training (up to step 2499), both training and validation perplexity gradually decreased, indicating that the model was learning and improving its ability to predict the next characters in the sequence. By step 2499, the training perplexity dropped to 27.4245, and validation perplexity to 28.3654. However, even at the end of the training, the validation perplexity remained above 28, suggesting that while the model has improved, it still possesses uncertainty in its predictions.

Initial Evaluation using cross-entropy loss (10M parameter)

```
10.795841 M parameters
step 0: train loss 4.0979, val loss 4.1011
step 500: train loss 3.3162, val loss 3.3529
step 1000: train loss 3.3113, val loss 3.3502
step 1500: train loss 3.3092, val loss 3.3478
step 2000: train loss 3.3093, val loss 3.3476
step 2499: train loss 3.3105, val loss 3.3462


e
mF
mt NlotayoL
nyGaaomon c:n
dr;,p oiveU,  nneltY
 eort t cp
;saiiTrLuurto diI?&ayoaoaiy t
-swnw, syr G
rresotstYg .o saeksMspg
 mhlsi raU s,o ltor t,l me nmritralrl
aaetedsIg
a
  ghlo ,qrl:DmTagehdtgnSep tebusat
dmtaese u,nhc ho  to
  QCdlByateayithv,i
 aes eTltn b?manty h es:n Se.edh hy
nIthth eXy c
oot lsp rto r.. htrataheueesEoa
yd cv    mseav e hrUsAspIsssdk G,,swfetpo ooa u tVdn'na ttnt  an non
sn AtRwFegnar rcitb wd  roY
jte.a aood ys.trd
nsd;mue aea Ctriweeystu;hnUr l lvwtmrev
lau ee
```

Evaluation using cross-entropy and perplexity (10M parameter)

```
10.795841 M parameters
step 0: train loss 4.1313, val loss 4.1500, train perplexity 62.2568, val perplexity 63.43391418457031:.4f
step 500: train loss 3.3165, val loss 3.3538, train perplexity 27.5631, val perplexity 28.610515594482422:.4f
step 1000: train loss 3.3112, val loss 3.3511, train perplexity 27.4188, val perplexity 28.532913208007812:.4f
step 1500: train loss 3.3088, val loss 3.3531, train perplexity 27.3512, val perplexity 28.591333389282227:.4f
step 2000: train loss 3.3130, val loss 3.3489, train perplexity 27.4675, val perplexity 28.470735549926758:.4f
step 2499: train loss 3.3114, val loss 3.3452, train perplexity 27.4245, val perplexity 28.36541175842285:.4f

h3oAswpBwey t a uhv , as ihffyom e'
enHS V  vt? hoerdHr
bnoey nsiU,y,Lee oaoeSsaA eoyevgeawec ltslnrr
rtfg oos h- l
pyudNi
eoIsi,ethn olNtrde ds wlhnae:sll
 e oU,ythwr in
 nskdr.enfs,
oo ytatarl
 tofihyrttii eioIdMRRefilstDtealso  cn m  sostron,
gitfshWtRcaenenpwr ongulormser h ds ,sfnaeoe:e aecT
ws mutH sd!aer e; eioO sa sriorMLe oiaedayuKmo iRaala nw f .y ne roa ly hun.aa-holtedhsryb eo,'h,
rsQe MwWc  FmF; da lptb ou  H tnli
  sims hoamor uilhn
inrEhKtttetIwdDter
Ouegt
NNeltt hc ilruEI;i
dtim
```

Whereas, to improve the model's performance, there are several strategies and techniques that can be utilized. Addressing these issues often involves a combination of modifying the architecture, tuning hyperparameters, revising learning rate and optimization algorithm, and refining the training process.

8. Conclusion

The model for next token prediction using the "tiny_shakespeare" dataset focused solely on the Masked Multi-Head Attention mechanism, the model architecture featured 6 layers with 6 attention heads and an embedding size of 384. Training involved a batch size of 64 and block size of 256, with training monitored via cross-entropy loss and perplexity. Despite initial instability with 10

million parameters, adjustments increased model complexity, leading to computational challenges such as overheating and memory exhaustion on both a laptop GPU and Google colab. These experiences highlighted the delicate balance between model complexity and computational feasibility, underscoring the need for robust hardware or cloud computing to manage advanced models effectively.

9. References

1. https://raw.githubusercontent.com/data/tinyshakespeare
2. https://www.3blue1brown.com/topics/neural-networks
3. https://proceedings.neurips.cc/files/paper/2017/Attention-is-all-you-need