Research Question: What impact do different layer normalisation methods (Post-LN vs. Post-PN vs. Post-BLN) have on the training effectiveness and performance of decoder-only transformer architectures (e.g. GPT-2)?

| (100 words)  **Introduction - Deep Learning and Decoder-only Transformers**    Deep-learning (DL), a specialised subset of machine-learning (ML), relies on sophisticated optimizer algorithms to discern complex patterns upon training on vast datasets. Unlike traditional ML which utilises simpler architectures, DL employs more layers of neural networks, enabling it to capture and process even more complex data structures with minimal human intervention.  A recent DL architecture—decoder-only transformers, had been profoundly important for the domain of NLP (natural-language-processing)—notably led to the creation of sophisticated chatbots like ChatGPT. Captivated by its inherent intelligence, my research aims to investigate potential architectural improvements in transformers, potentially improving its performance and reducing training cost. |
| --- |
| (400 words)  **Background Information - How decoder-only transformers work**  *Tokenizer and Positional Embedding (52 words)*  The diagram illustrates the tokenization process where text is converted into discrete tokens, each representing a piece of the original input. Following tokenization, positional embeddings are added, which encode the position of each token in the sequence. This allows the transformer to maintain sequence order, a vital aspect of understanding language structures.  *Q, K, V, and Self-Attention (68 words)*    Attention in transformers are calculated following:  Query, Key, Value matrices are derived from multiplying input embeddings by learned matrices. The dot products are scaled by ​​ to ensure numerical stability during learning. Softmax converts these values into a probability distribution, effectively selecting the input parts to focus on.  This operation basically tells the model to pay attention to more contextually significant words. For example, in *"the quick brown fox,"* higher weights from on 'fox' help the model emphasise fox’s when processing.  *Non-linear Activation Functions (50 words)*  The graph shows various non-linear activation functions such as ReLU and Sigmoid. These functions introduce non-linearity to the model’s decision function, allowing it to learn complex patterns in the data. Each activation function has a unique curve and activation threshold, providing different ways to activate neurons during the training process.  *Feedforward Networks (54 words)*  The block diagram outlines the structure of the feedforward layers in a transformer. Post-attention, the processed data passes through these layers, which apply further transformations to refine the model’s output. Each layer is fully connected and operates on the principle of transforming input features into higher-level representations before passing them to the next layer.  *Layer Normalisation (56 words)*  The diagram illustrates layer normalization which standardizes the inputs across the features within a layer. By adjusting and scaling the inputs, layer normalization helps in stabilizing the neural network’s training. It is crucial for combating the internal covariate shift, ensuring that each layer receives data within a scale that prevents the vanishing or exploding gradient problem.  *Backpropagation and Optimizers (63 words)*  The flowchart shows the backpropagation process where the model adjusts its weights based on the error gradient of the output. Accompanying this, a table of different optimizers like SGD and Adam highlights how these algorithms help minimize the loss function. Optimizers vary in how they adjust learning rates and handle gradient descent, directly impacting the convergence speed and stability of the model’s training. |
| (175 words)  Background Information - Layer Normalisation   * why LN needed * brief history on CNN + RNNs + BN (very brief only words) |
| (1000 words)  Theoretical Background - LN   * what does LN do (maths stuff begins)   + im expanding upon the work of BLN, but applying PN instead * RMSnorm vs PN vs LN vs BLN/PLN (m)   + explain their differences   + show architecture, differences   >> Formulates hypothesis here, based on differences, etc. |
|  |
| (500 words)  Experimental methodology and materials   * GPUs → Ubuntu, Nvidia A100, 96x-AMD-CPUs * PyTorch, implemented GPT-2 based on Karparthy’s course   + my modifications, using GPT-4 tokenizer instead. why? (150 words)   + torch.compile explanation ← speeding up (50 words?) ++ FLASH attention, CUDA speed up   + gradient checkpointing, HF   + Datasets → fineweb.edu     - upload to HF   + model sizes (n-embd) 128m, 753m, 1.5B   + sequence length – 1024 to 4096 * Implemented layer normalisation by code, and using sIncerass’ code, and facebook’s RMS code, and personal implementation of BLN (substituting) |
| Experimental Results and Reflection |
| (1550 words)  Analysis   * Evaluating metrics   + BLEU,   + hellaswag?   + fine-tuning for a few tasks * Raw Result Analysis * Experimental Analysis and Limitations * Conclusion |
| (275 words)  Evaluation and Improvements   * Further improvements   + larger models, more variations, larger token context   + better fine-tuning ?   + multi-modal ? (a lot more compute will be needed) * Evaluation   + better fine-tuning techniques probably will cause more impacts   + probably futile to be doing little optimisations like this? as better to instead wait for even stronger compute, allowing for newer architectures to be used |
|  |