Altering and Evaluating GPT-Based Text Generation

August 14, 2024

Alec Shamula, Akhil Gupta ashamul1@jhu.edu, agupt126@jhu.edu
Oluwatobi Ajide, Solomon Gruse oajide2@jhu.edu, sgruse1@jhu.edu
Engineering for Professionals, Student,
Johns Hopkins University
3400 N Charles St.
Baltimore, Maryland, USA

Abstract

The development of Generative Pre-trained Transformers (GPT) has significantly advanced the field of natural language processing, particularly in the domain of text generation (Kalyan, Rajasekharan, and Sangeetha, 2021). Building on this foundation, we introduce a series of novel GPT-based models designed to enhance text generation by incorporating advanced techniques for uncertainty quantification, prediction flexibility, and overconfidence mitigation.

We present Smooth-GPT, a model that utilizes label smoothing to temper overconfident predictions, promoting more generalized and balanced outputs. Bayes-GPT extends the traditional GPT architecture by integrating a Bayesian linear layer, enabling the model to capture and express uncertainty in its predictions. Sig-GPT replaces the conventional SoftMax activation with a Sigmoid layer, allowing for multi-label predictions and greater flexibility in generating diverse outputs. Finally, Bayes Sig-GPT combines the strengths of both Bayesian inference and Sigmoid activation, offering a unique approach to balancing uncertainty and prediction diversity.

These models aim to find more robust and adaptable language models, capable of generating high-quality, contextually appropriate text. This paper details the design and implementation of these models, demonstrating their potential to advance current text generation capabilities. Additionally, we discuss some initial performance evaluation results to provide a checkpoint and insights for future enhancements.

1 Introduction

Transformer-based models, particularly those following the Generative Pre-trained Transformer (GPT) architecture, have become the cornerstone of modern natural language processing, enabling impressive advancements in tasks such as text generation, machine translation, and summarization Schneppat (2023). Despite their widespread success, there remains significant room for innovation in how these models handle uncertainty, generate diverse outputs, and avoid overconfident predictions.

Standard Large Language Model (LLM) architectures employ a SoftMax activation function to the network's output logits Dipan (2023). In doing so, the raw logits are transformed into a probability distribution across all possible tokens in the vocabulary, enabling the next-token-generation task. However, given the inherent diversity of language, this approach overly penalizes "incorrect" token generations during training. For example, the semantic meaning of "the man walked to the store" is just as effectively conveyed with "the man walked to the shop." This problem is mitigated with a large enough training corpus that better captures linguistic diversity.

This paper introduces a series of novel GPT-based models, each designed to enhance text generation through different techniques. Smooth-GPT leverages label smoothing to reduce overconfidence in model predictions, leading to outputs that are more balanced and less prone to over-fitting. Bayes-GPT introduces a Bayesian linear layer into the GPT architecture, enabling the model to learn a distribution over its weights and, consequently, to quantify uncertainty in its predictions. Sig-GPT departs from the traditional SoftMax activation by implementing a Sigmoid layer, which allows for multi-label prediction and supports the generation of more varied and flexible text outputs. Finally, Bayes Sig-GPT combines the Bayesian inference approach with Sigmoid activation, offering a unique solution that balances the need for uncertainty quantification with the ability to produce diverse and contextually appropriate text.

By incorporating techniques like label smoothing, Bayesian inference, and multi-label prediction, we aim to push the boundaries of what is possible in text generation. This paper explores the design and implementation of these models, illustrating how they can be used for text generation and offers an evaluation on the current state of each of these models.

2 Approach

2.1 Dataset

Each model is trained and validated using the WikiText103-v1 dataset provided by Salesforce (2023). The dataset includes over 100 million tokens. Tokens are extracted from a collection of verified Wikipedia articles.

2.2 Hardware

CUDA-enabled GPUS (A100, H100) were utilized to power the training of models. Models were capped at a time limit of 72 hours to train.

2.3 Baseline Model

To gauge performance against each proposed method, a standard LLM with SoftMax output activation is trained. This model uses a GPT-2 style architecture with approximately 124 million parameters. The architecture uses 12 layers with 12 attention heads per transformer layer. The resulting embedding layer is a vector of length 768. The baseline code was provided by Andrej Karpathy's nanoGPT GitHub repository (Karpathy, 2022).

During training, a block size of 512 is used, alongside a batch size of 12. The model was trained for 100,000 iterations. The general architecture, as well as training configuration was kept consistent for each model. However, due to increased compute requirements of the Sigmoid-based models, the number of training iterations for this set of models was significantly less. This is discussed in more detail in the sections that follow.

2.4 Smooth-GPT

The first model variation applies label smoothing to the baseline GPT-2 model's training scheme. Contrary to an architectural change, this approach modifies the output values in which Cross Entropy Loss is computed against. The neural network architecture of this model is identical to that of the Baseline. Smooth-GPT was also trained for 100,000 iterations. With label smoothing, however, the target vector for each input sequence is no longer strictly binary. That is, "incorrect" tokens are given a small value. This value is computed with respect to the smoothing factor ϵ and the size of the vocabulary V. In the case of GPT-2, the vocabulary size is 50,257 tokens. In addition to "smoothing out" all tokens in the vocabulary, the target token's value is reduced by ϵ. Equation (1) shows how the updated target vector is created.

$$y_i = \begin{cases} 1.0 - \epsilon & \text{if } i = \text{correct token} \\ \frac{\epsilon}{V-1} & \text{if } i \neq \text{incorrect token} \end{cases} \tag{1}$$

Intuitively, this translates to each token in the vocabulary having a small probability of being the "correct" token. See Figure 1 for a graphical representation of this. Since the original target token still has the greatest probability, label smoothing does not cause significant degradation in performance. Rather, this approach may improve model generalization with its inherent regularization and "built in" uncertainty. The hypothesis is that this makes the LLM more resilient to noisy examples. In addition, by distributing probability mass to other tokens, the model is less likely to repeatedly sample the same tokens. This suggests that the model will produce more creative text.

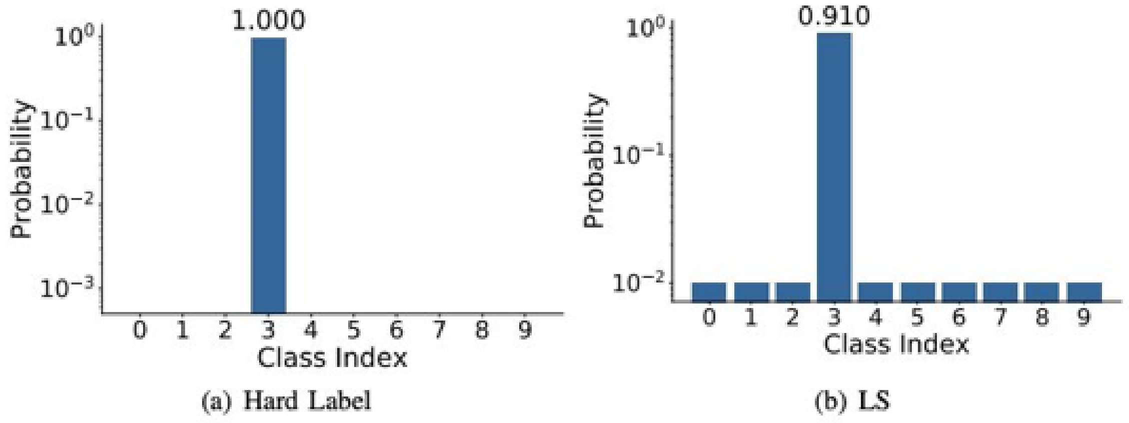

Figure 1: The target probabilities for each token (class) as used by the baseline model (left) vs. the target probabilities for each token as used by the label smoothing model (right).

2.5 Bayes-GPT

While the label smoothing model built uncertainty into the target vector, this research also poses a means of learning uncertainty directly. This is accomplished by replacing the ordinary linear output layer with a *Bayesian* linear output layer. In doing so, the model learns a distribution over the output layer's weights. This is demonstrated by an example Bayesian Network in Figure 2. While only a single layer is modified, it is worth noting that this variation introduces an increase in the overall number of model parameters. Each network connection has an associated mean and standard deviation, as opposed to a static weight. The resulting model thus has approximately 160 million parameters. Recall that the baseline GPT-2 model has 124 million parameters. As with the Baseline, Bayes-GPT was trained for 100,000 iterations.

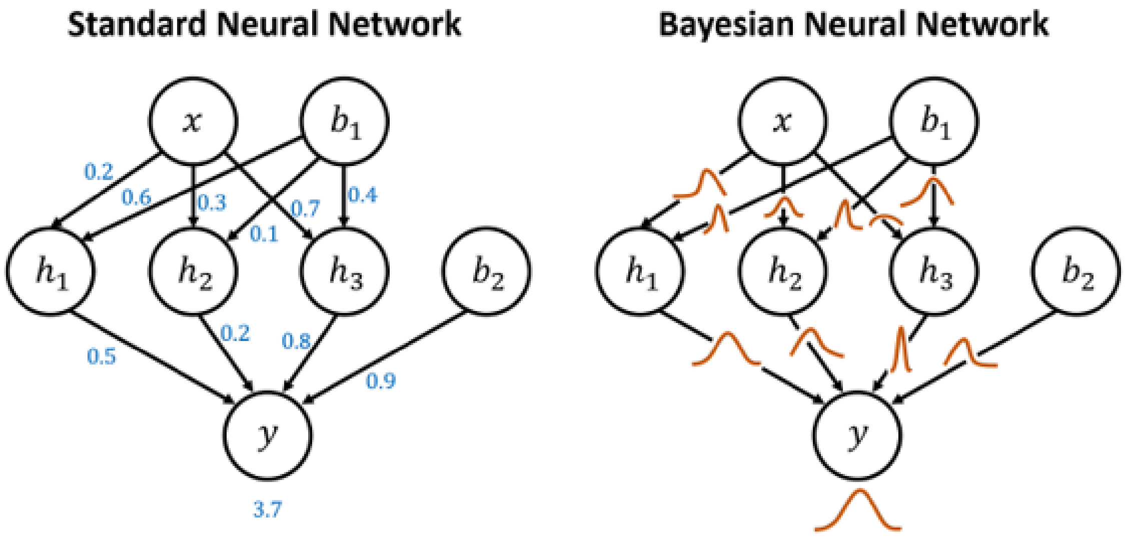

Figure 2: Standard neural network with static weights (left) vs. Bayesian neural network (right).

By introducing a Bayesian layer, the model attempts to learn the posterior distribution of weights, $P(W|X)$, where W is the set of model weights and X is the training data. Since learning $P(W|X)$ is often computationally infeasible, Bayesian Neural Networks approximate it using a simpler, parametric distribution $Q(W)$. Oftentimes, the Gaussian distribution is used, as is the case in this research. During the learning process, the objective is to minimize the Kullback-Leibler (KL) Divergence between $P(W|X)$ and $Q(W)$. Since $P(W|X)$ is intractable, we can estimate this term using the Evidence Lower Bound (ELBO). This approach approximates the posterior distribution using the Cross Entropy loss function with an additional KL Divergence term between $Q(W)$ and the prior $P(W)$. The objective function is shown in Equation (2). The Cross Entropy term seeks to ensure predictions are accurate given the training data (as is done in standard LLM training). The

KL Divergence term measures the difference between the learned weight distribution and the prior distribution.

$$L = \sum_{i=1}^{N} y_i logp(y_i|x_i, \theta) + \beta KL(q(\theta)||p(\theta)) \tag{2}$$

At training time, we use Variational Inference to approximate the distribution $Q(W)$. This is done using a single point estimate over many iterations. This approach differs from other techniques such as Markov Chain Monte Carlo (MCMC) that would sample multiple times across the weights to get a more accurate approximation of the posterior distribution. At inference time, weights are sampled from this distribution and are used to produce a point estimate (to generate text). In addition, the weight distributions create a measure of uncertainty for the generated text.

2.6 Sig-GPT

Sig-GPT presents a different technique for capturing uncertainty in language modeling. This approach frames the text generation task as a multi-label problem. This is accomplished by replacing the SoftMax activation function in the output layer with the Sigmoid function. With this framework, the output probabilities are computed independently of one another, enabling multiple tokens to be considered “correct” during each generation step.

With the inclusion of the Sigmoid output activation function, the ordinary Cross Entropy loss function is no longer viable. Instead, Binary Cross Entropy is used (Equation (3)). This objective function evaluates the output probability for each token independently. Let p_i and t_i be the Sigmoid probabilities produced by the model and ground truth labels, respectively, for token i in the vocabulary.

$$L_{BCE} = -\frac{1}{N} \sum_{i=0}^{N} t_i log(p_i) + (1 - t_i) log(1 - p_i) \tag{3}$$

This architecture is enabled by employing a data augmentation module. For any given sequence, m indices are selected at random. Each index corresponds to a token in the sequence. At each index, k candidate tokens are generated based on the output from a pretrained GPT-2 model. These candidates are generated via the top-k sampling strategy. Candidates are filtered based on a perplexity threshold. The resulting target vector is a binary array of shape (B, N, V) where B is batch size, N is block size, and V is vocabulary size. The complete Data Augmenter architecture is shown in Figure 3.

It is worth noting that due to the additional computational overhead of applying the Data Augmenter, Sig-GPT and its variants were only trained for 8,000 iterations each.

2.7 Sig-GPT Regularization

In addition to the baseline Sig-GPT, we propose two additional regularization techniques. The first approach includes an additional squared error term in conjunction with the Binary Cross Entropy loss. The model is penalized if the sum of the probabilities across the Sigmoid outputs far exceeds a value of 1. This term encourages the model to balance maintaining something that nearly represents a probability distribution over all the tokens while also

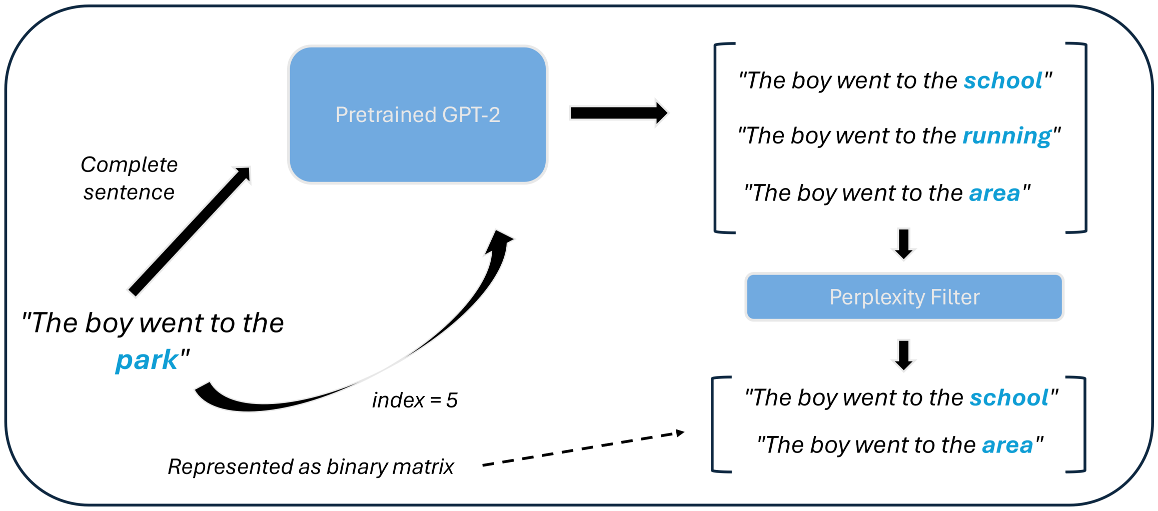
Pretrained GPT-2
Complete sentence
"The boy went to the park"
index = 5
Represented as binary matrix
"The boy went to the school"
"The boy went to the running"
"The boy went to the area"
Perplexity Filter
"The boy went to the school"
"The boy went to the area"

Figure 3: Data Augmenter architecture diagram.

providing it the flexibility to assign multiple tokens high probability values as it sees fit. This loss can be expressed as follows:

$$L = L_{CE} + ((\sum_{i=1}^{V} p(y_i)) - 1)^2 \tag{4}$$

The other method of regularization that we use is positive class weighting. This approach adds an additional weight to the positive class when calculating the Binary Cross Entropy loss.

$$L = -\frac{1}{N}\sum_{i=1}^{N} w y_i log p(y_i|x_i,\theta) + (1-y_i)log(1-p(y_i|x_i,\theta)) \tag{5}$$

The motivation for introducing this regularization term is due to the imbalance between the positive and negative class. The model will more frequently encounter predictions where most of the tokens in the vocabulary are set to 0. We want to increase the loss contribution from the positive examples to prioritize correct predictions of positive labels. This decision can be related back to the Precision-Recall trade off. Precision captures the percentage of properly classified positive instances from all instances that the model predicted as positive. This is denoted as:

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

Recall, on the other hand, quantifies the percentage of positive instances that the model was able to correctly identify. Recall is defined as follows:

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

By introducing a weighting on the positive class, the model is rewarded for prioritizing defections of positive instances. The is synonymous with increasing Recall.

2.8 Bayes Sig-GPT

The final GPT-variation evaluated is Bayes Sig-GPT. This architecture combines the modifications of Bayes-GPT and Sig-GPT into a single model. As with Bayes-GPT, a Bayesian linear layer replaces the standard linear layer prior to the network's output. In addition, the loss function is modified to be Binary Cross Entropy with an added KL Divergence term. As with Sig-GPT, the Data Augmenter is used with this approach. Due to the larger model size and data augmentation required, time restrictions caused this model to only be trained for 6,000 iterations.

3 Results

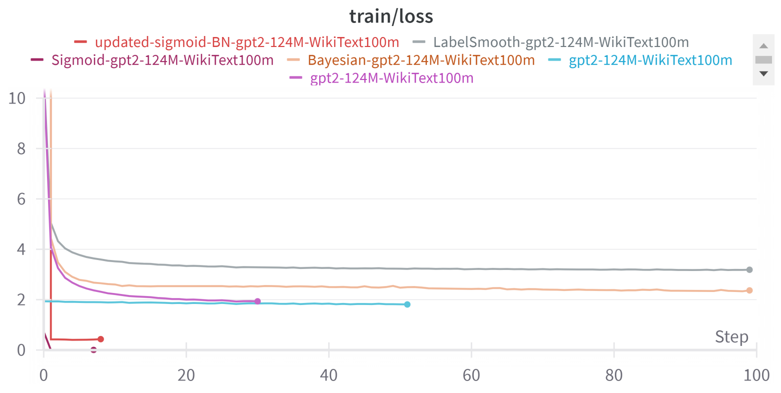

Figure 4: WANDB Training Loss Visualization

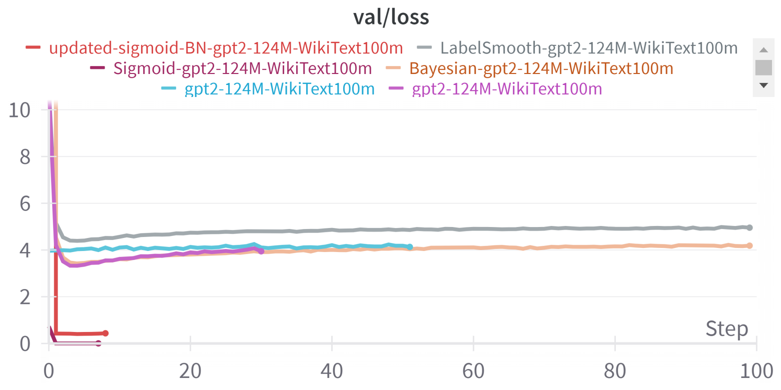

Figure 5: WANDB Validation Loss Visualization

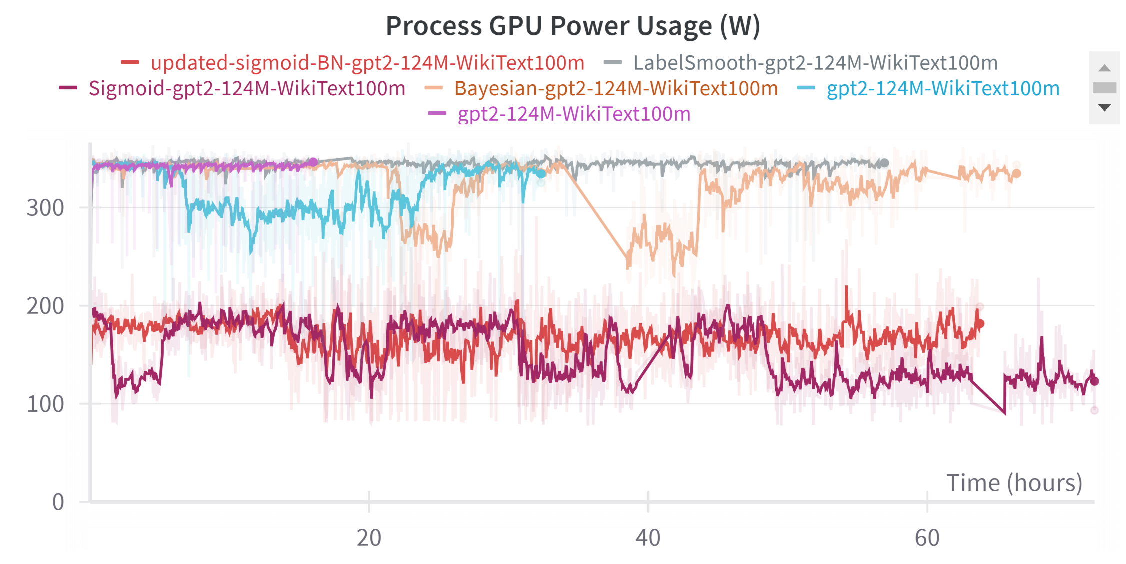

Figure 6: WANDB GPU Utilization

	Perplexity	Self-BLEU	Distinct-4
Baseline	6.1855	27.1660	0.9709
Smooth-GPT (ε = .05)	6.6225	26.6854	0.9743
Smooth-GPT (ε = .1)	7.2704	18.3164	0.9901
Bayes-GPT	10.0340	28.3591	0.9641
Sig-GPT (Baseline)	2134.7043	18.5969	0.9907
Sig-GPT (Regularized)	50259.5078	16.4033	0.9901
Sig-GPT (PW)	2028.1080	18.2633	0.9907
Bayes Sig-GPT (PW)	2257.3240	18.5297	0.9907

Table 1: Comparison of models based on Perplexity, Self-BLEU, and Distinct-4.

Experiment results provide valuable insights into the performance and diversity of the tested models, as summarized below.

3.1 Wandb Training Visualizations

Training was piped into the Weights Biases (Wandb) machine learning operations visualization platform. We can see that most of our models have converged, however there are evidently varying scales of loss in Figure 4 and 5 which result from the usage of different inherent loss functions. The added KL divergence term raises the Bayes-GPT loss, while the nature of Sig-GPT's binary cross entropy loss function brings it down. This difference in scale inhibits the ability to draw any useful comparison insight from the loss charts. In Figure 6, we can see a remarkably lower GPU power utilization by the Sigmoid based models. This is assumed to be the cause for lower efficient training of Sig-GPT and may be accredited to the lack of CUDA-enabled code for the data augmentation mechanism of the architecture. Regardless, this signifies an area for improvement on Sig-GPT.

3.2 Perplexity

Perplexity measures the effectiveness of a model in predicting the next word, with lower values indicating better performance. The Baseline model achieves the lowest perplexity score of 6.1855, suggesting it has the best predictive accuracy among the models evaluated. In contrast, the models incorporating Sig-GPT, especially the Baseline and PW variants, show significantly higher perplexity scores, with values reaching up to 2134.7043. These elevated perplexity scores indicate that these models may struggle with accurate predictions, potentially due to issues such as over-fitting or sub-optimal parameter settings.

3.3 Self-BLEU

Self-BLEU assesses the diversity of the generated text by measuring how similar the generated samples are to each other. A lower Self-BLEU score signifies greater diversity. The Smooth-GPT model with $\varepsilon = 0.1$ achieves the lowest Self-BLEU score of 18.3164, indicating the highest diversity among the models tested. Conversely, Bayes-GPT has a higher Self-BLEU score of 28.3591, suggesting reduced diversity. The Sig-GPT variants also show high diversity, with Self-BLEU scores ranging between 16.4033 and 18.5969.

3.4 Distinct-4

Distinct-4 evaluates the diversity at the 4-gram level, with higher values indicating more unique sequences. The Sig-GPT models (Baseline, Regularized, and PW) and Bayes Sig-GPT (PW) achieve the highest Distinct-4 scores, all around 0.9907. This indicates that these models generate a wide variety of unique 4-grams, reflecting their ability to produce diverse and less repetitive text. The Baseline model also shows strong performance with a Distinct-4 score of 0.9709, although it is slightly lower than the Sig-GPT variants.

3.5 Summary

The results highlight a trade-off between prediction accuracy and text diversity. The Baseline model excels in predictive accuracy, as indicated by its low perplexity, but it does not achieve the highest diversity. In contrast, the Sig-GPT models, despite their high perplexity scores, demonstrate high levels of text diversity in terms of Self-BLEU and Distinct-4. This suggests that while these models may produce more varied and less repetitive text, they do so at the expense of prediction accuracy. These findings emphasize the need to balance these aspects depending on the specific goals of the application.

4 Conclusion

The results of this study were inconclusive, primarily due to limitations in both model size and dataset size. The base GPT-2 model has just 124 million parameters, which is significantly smaller than today's state-of-the-art (SoTA) LLMs. Additionally, the WikiText dataset used for training is also fairly small relative to SoTA LLM training. As a result, even the baseline model was unable to maintain consistent themes and thoughts while generating text. Oftentimes, the model regurgitated Wikipedia-like text (similar to the training data). This made analysis more difficult. Future research would benefit from a larger model and more robust dataset (more than just Wikipedia articles). This would likely show a greater impact by each of the proposed architectures.

Regarding the label smoothing approach, in-depth hyper-parameter tuning is required to better understand the impact of this method. Specifically, additional α values should be explored. This is fairly feasible since this design did not introduce any additional computational overhead. The Sig-GPT model, on the other hand, required a more complex training procedure. This was largely due to the computational requirements of the Data Augmenter. Given additional training time, the performance of the Sigmoid-style models can be more properly evaluated. Finally, due to the added parameters, Bayes-GPT was slightly slower and less memory efficient than the baseline approach.

While the results of this study are ultimately inclusive, they are intended to serve as a starting point for further research and development in the field of text generation ambiguity and diversity in LLMs.

References

Dipan. Why do we use Softmax in Transformers? Medium, October 2023. URL `https://medium.com/@maitydi567/why-do-we-use-softmax-in-transformers-fdfd50f5f4c1`.

Katikapalli Subramanyam Kalyan, Ajit Rajasekharan, and Sivanesan Sangeetha. Ammus : A survey of transformer-based pretrained models in natural language processing, 2021. URL `https://arxiv.org/abs/2108.05542`.

Andrej Karpathy. nanogpt github repository, Dec 2022. URL `https://github.com/karpathy/nanoGPT?tab=readme-ov-file`.

Salesforce. Wikitext-103-v1. `https://huggingface.co/datasets/Salesforce/wikitext#wikitext-103-v1`, 2023. [Data set].

Jo Schneppat. Gpt: Transformer model. *Schneppat AI*, 2023. URL `https://schneppat.com/gpt-transformer-model.html`.