HW4 Part 2 Report - STATS305B

Junze (Tony) Ye
Stanford University
Stanford, CA 94305
`junze@stanford.edu`

Kantapong Kotchum
Stanford University
Stanford, CA 94305
`kkotchum@stanford.edu`

Abstract

Telltale signs of semantic deficiency in code made by the baseline transformer model and evidence of overfitting motivate our group to make architectural modifications. We posit that the first defect can be reduced by enhancing the transformer model's ability to learn the nonlinear dependencies across the embedding space, whereas the second defect is due to the lack of parameter regularization. Guided by these hypotheses, we propose and implement changes to how the hidden layer is activated, as well as adding dropout layers to both the attention heads and the MLP. Multiple computational results reveal little change to the modified transformer model's performance nor the quality of its generated code. Our findings suggest the relative robustness of the transformer model to architectural revisions. Moreover, it foregrounds the value of utilizing more compact architectures like SSMs when the available compute resource is constrained.

1 Introduction

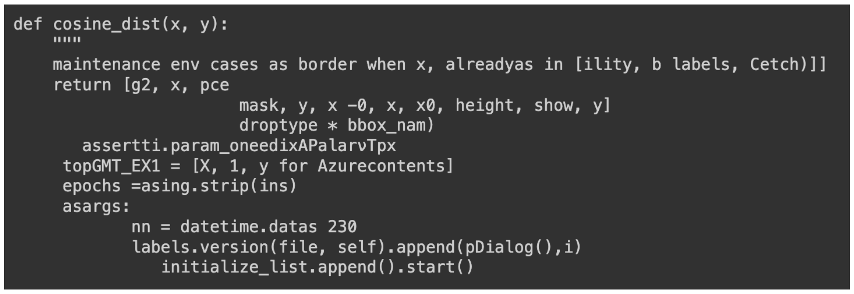
Since its inception, the transformer architecture has revolutionized Natural Language Processing (NLP) and the broader field of sequential data modeling, allowing contextualization of nearby tokens for various tasks such as text generation and machine translation [Vaswani et al., 2017]. Part 1 of HW4 saw our group's implementation of a decoder-only transformer LM capable of completing code snippets. The training results of Part 1 inspire our attempts to revise the architecture design. In the following subsections, we discuss two motivating observations from the training results.

1.1 Semantic Deficiency

While syntactic structures of text generations in HW4 Part 1 are highly similar to real code, their **semantic** content is far from being satisfactory. For example, in Figure 1, although the generated code has reasonable line lengths and proper indentations, the variable and function names bear little relevance to the prompting text, which wants to define a function to calculate the cosine distance.

We postulate that the constrastive performance of syntactic vs semantic learning can be explained by the two-stage framework of transformer learning outlined in Turner [2023]. To summarize Turner's interpretation, we can view a decoder-only transformer block as two consecutive stages of representation learning. The first stage operates **"across the sequence"**, i.e., it aggregates sequence-wide information by taking a weighted average across T token embedding vectors. Although attention weights are a nonlinear function of embedding vectors, the weighted average is still a linear combination. Nonlinearity comes in the second learning stage, which operates **"across the features"** and mixes the D features of each token embedding in a nonlinear manner; a token embedding, already attention-weighted in the first stage, is passed through a residually connected MLP sub-block to capture the nonlinear dependencies between the features of a token embedding.

Since the activation function is the main ingredient introducing nonlinearity to the architecture, we hypothesize that its functional choice plays a non-trivial role in the model performance. The skeleton



```
def cosine_dist(x, y):
    """
    maintenance env cases as border when x, alreadyas in [ility, b labels, Cetch)]]
    return [g2, x, pce
                       mask, y, x -0, x, x0, height, show, y]
                       droptype ∗ bbox_nam)
       assertti.param_oneedixAPalarvTpx
     topGMT_EX1 = [X, 1, y for Azurecontents]
     epochs =asing.strip(ins)
     asargs:
            nn = datetime.datas 230
            labels.version(file, self).append(pDialog(),i)
               initialize_list.append().start()
```

Figure 1: Sample generated by the transformer LM in HW4 Part 1 (ReLU, no Dropout).

code provided for HW4 uses the ReLU function as the activation for the MLP sub-block. However, we note that popular transformer-based models like GPT and BERT use the GELU (Gaussian Error Linear Unit) activation, as introduced in Hendrycks and Gimpel [2016]. GELU smoothly approximates the ReLU function with non-zero gradients for negative inputs, which can help in reducing the dead neuron problem associated with ReLU and potentially lead to better learning dynamics. Since maintaining gradient flow is crucial in the training of transformer models, we are motivated to verify the potential of GELU to improve the semantic quality of the model outputs.

1.2 Model Overfitting

In our experiments, the validation loss becomes 30% higher than training loss after 1800 iterations. When we increase the number of training iterations to 4000, we notice that eventually the validation loss flatlines to 2.9 before increasing back up again as training loss continues decreasing below 2. The loss dynamics suggests that our model is overfitting to the training data. The overfitting phenomenon is somewhat unexpected because one would agree that, at 36M parameters, our transformer LM is vastly under-parametrized compared to SOTA models like GPT-4.

Given the time and resource constraints, we acknowledge the difficulty of offering a theory-based explanation of why overfitting occurs in a transformer model. Instead, we focus on identifying ways to reduce the level of overfitting by computational experiments. For example, dropout is a common technique used in deep learning community to regularize the learning parameters and prevent overfitting [Hinton et al., 2012]. This is done by randomly selecting certain proportions of nodes (the proportion is often pre-specified as a hyperparameter) to be temporarily removed during training. When training complex deep architectures like transformers, practitioners often incorporate dropout for better generalization.

2 Hypothesis

In previous sections, we have discussed possible implications of using ReLU activation in the MLP sub-block and adding dropout layers to reduce over-fitting during model training. We crystallize our intuition into the following two hypotheses, implement model changes suggested by the hypotheses, and run experiments to test their validity. The hypotheses are:

1. The smoothness of GELU allow it to facilitate better gradient propagation compared to ReLU, leading to better learning across feature dimensions. Hence, replacing ReLU by GELU as the activation function for the MLP sub-block of a transformer model would improve the model performance and in particular the problem of semantic deficiency.
2. Regularizing the transformer model by dropouts will enable better generalization and reduce the overfitting phenomena we observed in HW4 Part 1.

3 Method

Smoothing the nonlinearity: To test our hypothesis, the ReLU activation function originally used in the MLP of the transformer is replaced by GELU activation function. We do not add the dropout layer yet in this experiment to avoid confounding factors.

Dropout regularization: Let `H`, `T` and `D` be the number of heads in multi-head attention, context window size, and token embedding dimensions, respectively. For the dropout implementation, dropout layers with probability $p = 0.15$ are introduced in three places of a transformer block:

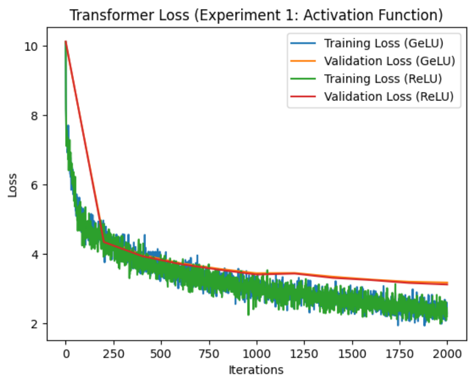
1. When generating individual attention matrices in a forward pass of HEAD, we zero out a portion of the attention weights.
2. As an output of MULTIHEADATTENTION, when projecting stacked token embedding outputs from all heads of dimension `HxTxT` into a single embedding output of dimension `TxT`.
3. When passing output of the MULTIHEADATTENTION into the FEEDFORWARD MLP.

For each hyperparameter setting, we ran training for 2000 iterations on a Tesla V100 GPU, learning rate at 10^{-4}, mini-batch size at 32, and evaluate the validation loss every 200 iterations.

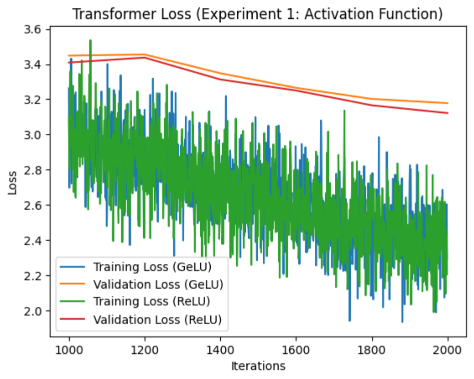
4 Result

4.1 Implementation 1: Varying the MLP Sub-block's Nonlinearity

Figure 2 are plots showing the train vs test losses of models with GELU and ReLU activation functions. The plot on the left tracks the whole training period of 2000 iterations and the other zooms in on the second half of the training session.



(a) 2000 Iterations



(b) Second Half

Figure 2: ReLU vs GELU Experiment (Train vs Validation).

We have also included in the Appendix section tables 1 and 2 for the final training and validation losses at 2000 iterations, as well as each model's training time.

4.2 Implementation 2: Adding Dropout Regularization

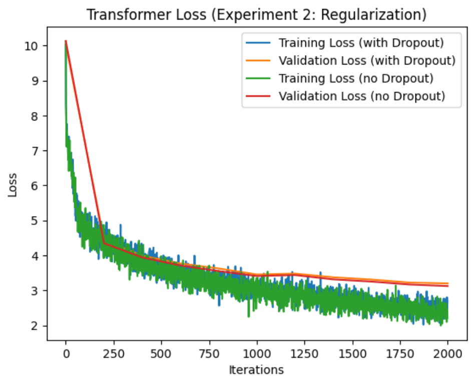
Figure 3 are loss curves for our experiments with adding dropout regularization.

4.3 Qualitative Evaluations

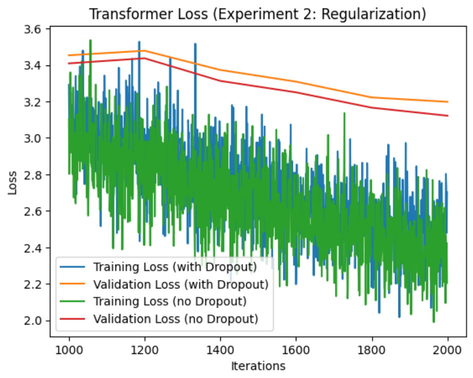
Figures 5 and 6 in the Appendix respectively showcase a sample text generated by a ReLU-activated transformer with dropout, and one generated by a GELU-activated transformer without dropout. Qualitatively speaking, code generated by the ReLU-activated transformer with dropout has on average shorter characters per line than does that produced by other model variants. However, we do not see any visible improvement of semantic content in all the model modifications we tried.

5 Discussion

Based on the experiment data we collected, we make the following observations:



(a) 2000 Iterations



(b) Second Half

Figure 3: Dropout Regularization Experiment (Train vs Validation).

1. Changing the nonlinearity by toggling the activation function neither reduces training speed nor improves the model performance in training and validation.
2. Dropout regularization not only slows down training, but the associated model also achieves slightly worse performances on both training and validation data.
3. Overall, there are no substantial differences between the baseline transformer, the GELU-activation transformer, and the transformer with dropout.

Results from the first experiment suggest that the original transformer model does not seem to have a gradient propagation issue in training. The evidence is inconclusive regarding if the nonlinear dependencies across the feature dimensions are adequately captured by the original transformer model.

Results from the second experiment indicate that we have not trained on enough data. In general, the effect of dropout regularization is salient only when trained on sufficiently large number of data. Inspecting the dataset, we discovered that the training corpus comprises of only 1172 code files. Moreover, the training corpus is not homogeneous, since some of the samples contain non-English tokens such as Chinese and Korean characters. As our transformer is under-parametrized and trained on a small dataset, it makes sense that it achieved lower performance when dropout is applied. Since dropout randomly removes neural network connections when our model is already under-parametrized, its capability to learn from data can only get worse.

All in all, these observations indicate that unless we switch to a more train-efficient and compact architecture like S4 or Mamba [Gu et al., 2021], achieving better model performance would require us to scale up both the transformer model size and training time to capture the complexity of programming texts.

6 Conclusion

In this work, we modified the transformer architecture implemented in HW4 Part 1 in two ways: using a smoother nonlinearity for the feedforward sub-block, and applying dropout regularization. Unfortunately, our computational results from both implementations showed little improvement in the model performance. We posit that this is due to limitations of our model's parameter size and constraints on our compute power, and highlight the need for a more compact and efficient architecture. Our code is available at https://github.com/junzeye/stats305b_LLM.

References

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Richard E Turner. An introduction to transformers. *arXiv preprint arXiv:2304.10557*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

7 Appendix

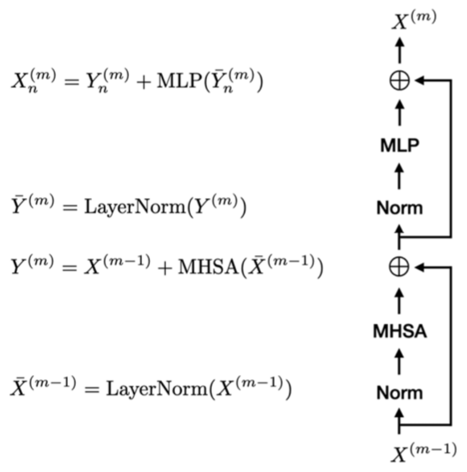

$X_n^{(m)} = Y_n^{(m)} + \mathrm{MLP}(\bar{Y}_n^{(m)})$
$\bar{Y}^{(m)} = \mathrm{LayerNorm}(Y^{(m)})$
$Y^{(m)} = X^{(m-1)} + \mathrm{MHSA}(\bar{X}^{(m-1)})$
$\bar{X}^{(m-1)} = \mathrm{LayerNorm}(X^{(m-1)})$
$X^{(m)}$
MLP
Norm
MHSA
Norm
$X^{(m-1)}$

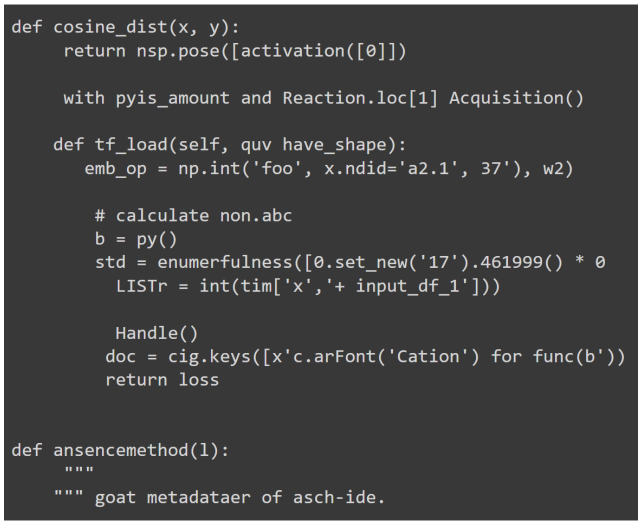
Figure 4: Transformer Block Diagram

Activation Function	Train Loss	Val Loss	Training Speed
ReLU (Baseline)	2.4911	3.1460	$\sim$ 7it / s
GELU	2.5045	3.1514	$\sim$ 7it / s

Table 1: Activation Function Comparison

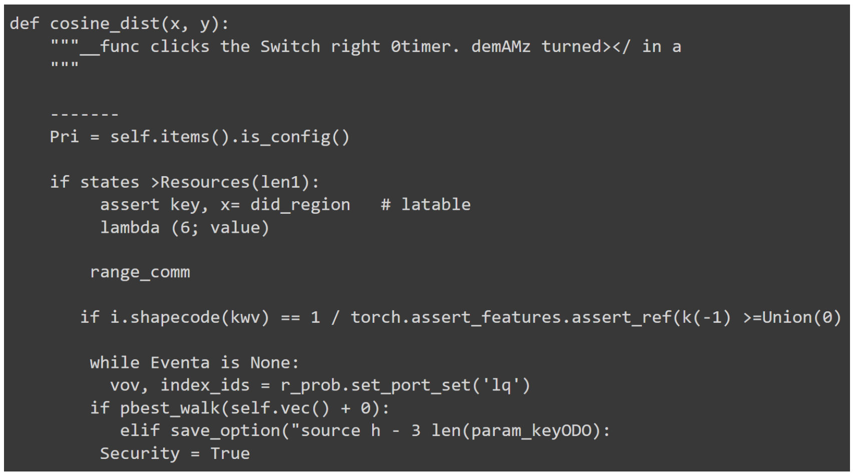
Activation Function	Train Loss	Val Loss	Training Speed
No Dropout (Baseline)	2.4911	3.1460	$\sim$ 7it / s
Dropout	2.5591	3.1979	$\sim$ 6it / s

Table 2: Dropout Comparison



```
def cosine_dist(x, y):
     return nsp.pose([activation([0]])

     with pyis_amount and Reaction.loc[1] Acquisition()

    def tf_load(self, quv have_shape):
       emb_op = np.int('foo', x.ndid='a2.1', 37'), w2)

        # calculate non.abc
        b = py()
        std = enumerfulness([0.set_new('17').461999() * 0
          LISTr = int(tim['x','+ input_df_1']))

          Handle()
         doc = cig.keys([x'c.arFont('Cation') for func(b'))
         return loss


def ansencemethod(l):
     """
    """ goat metadataer of asch-ide.
```

Figure 5: Sample code generated by the transformer LM in Part 2 (ReLU, with Dropout).



```
def cosine_dist(x, y):
    """__func clicks the Switch right 0timer. demAMz turned></ in a
    """

    -------
    Pri = self.items().is_config()

    if states >Resources(len1):
         assert key, x= did_region   # latable
         lambda (6; value)

        range_comm

       if i.shapecode(kwv) == 1 / torch.assert_features.assert_ref(k(-1) >=Union(0)

        while Eventa is None:
          vov, index_ids = r_prob.set_port_set('lq')
        if pbest_walk(self.vec() + 0):
           elif save_option("source h - 3 len(param_keyODO):
         Security = True
```

Figure 6: Sample code generated by the transformer LM in Part 2 (GELU, no Dropout).

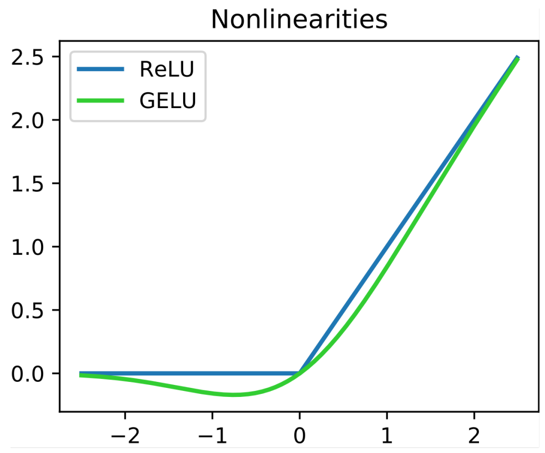


Figure 7: ReLU vs GELU activation functions.