PROJECT MID SUBMISSION REPORT

Team :

- Anushka Agrawal (2021101023)
- Vanshika Dhingra (2021101092)
- Khushi Wadhwa (2021101104)

Why do we do Pruning?

- Neural networks are often **over-parameterized**, meaning that they have more parameters than they actually need to learn the desired task. This means that many of the weights in the network are not actually doing anything useful. Pruning can help to remove these redundant weights, making the network more efficient.
- Neural networks are often trained using stochastic gradient descent (SGD). SGD can sometimes get stuck in local minima, which can lead to **suboptimal** performance. Pruning can help to prevent SGD from getting stuck in **local minima** by encouraging the network to learn more robust features.
- Pruning can help to improve the **generalization** performance of neural networks. Generalization performance is the ability of a network to perform well on unseen data. Pruning can help to improve generalization performance by removing weights that are not important for the specific task that the network is being trained on.

What does Pruning Do?

During pruning, connections associated with less important or redundant features are removed, leading to a sparser neural network. This can remove the connections based on various saliency criterion or some other methods. So

pruning kind of works like feature selection i.e. selecting features most important for the task.

How does retraining help?

During pruning since we remove the connections our accuracy will decrease. Now by retraining the network is encouraged to rely on the remaining important features and adjust their weights to capture the relevant patterns in the data.

What is the difference between structured and unstructured pruning?

- Structured pruning involves pruning entire neuron, filters etc.
- Unstructured Pruning involves pruning individual parameters based on some saliency criteria.

What we have done through our whole study is unstructured pruning

Progress So Far

Our Task

The task which we are working on is **Neural Machine Translation.** It involves translating a sentence X of one language(source language) to sentence y of another language(target language).

Encoder Decoder Architecture:

An encoder computes representations for each source sentence.Decoder generates one target word at a time.

Alignment: Word level correspondence between source sentence x and target sentence y.

What we calculate in **NMT** is:

$$p(y \mid x) = p(y_1 \mid s) \cdot p(y_2 \mid y_1, s) \cdot \ldots \cdot p(y_T \mid y_{T-1}, y_{T-2}, \ldots, y_1, s)$$

Beam Search Decoding: On each decoding step we keep track of k most probable translations (which we call hypothesis).

Word Embeddings: Word in sentence are converted into vectors such that these vectors are learned to capture context and similar words used in similar context have more close word embeddings. Basically we get the features of the words in the form of numbers.

LSTM with Attention

The **Long Short-Term Memory (LSTM)** model is a type of Recurrent Neural Network (RNN) designed to overcome the vanishing gradient problem and capture long-term dependencies in sequential data.

Architecture Overview

- **Encoder**: A stack of LSTM layers processes the input sentence word-by-word, producing hidden states at each time step.
- **Attention Mechanism**: Introduced to overcome LSTM's bottleneck of compressing the entire input into a single vector. At each decoder step, the model computes a **context vector** as a weighted sum of encoder hidden states, using attention scores.
- **Decoder**: Another LSTM stack generates the output sentence, conditioning on both the context vector and previous outputs.

Key Properties

- **Sequential Processing**: Tokens are processed one at a time in both encoder and decoder.
- **Temporal Dependency Modeling**: Excels at learning time-step dependencies, useful for language tasks.
- **Memory Cell**: Helps retain long-term information without being washed out.

Transformer Model

The **Transformer** architecture revolutionized NLP by replacing recurrence with **self-attention** mechanisms, enabling models to learn dependencies regardless of distance between tokens.

Architecture Overview

- **Encoder**:
  - Multi-head Self-Attention layers allow the model to look at all input positions simultaneously.
  - Feedforward layers apply transformations to the attention outputs.
- **Decoder**:
  - Similar architecture with additional **cross-attention** to incorporate encoder outputs.
  - Masked self-attention allows autoregressive generation.
- **Positional Encodings**: Since there's no recurrence, sinusoidal positional encodings are added to input embeddings to preserve order.

Key Properties

- **Parallelization**: Allows faster training due to non-sequential computation.
- **Multi-head Attention**: Learns diverse types of relationships between tokens.
- **Layer Normalization**: Helps with stable and deep training.

Evaluation Measures

1. **Perplexity:**

Intrinsic measurement of how good a model is. A good model is one which gives high probability to the correct sentence

Perplexity is exponential of average negative log likelihood. Decreasing perplexity is equivalent to increasing probability. Suppose we have 10 items with equal probability then perplexity is 10 showing that the model cannot choose among those 10 and is confused.

$$Perplexity = \frac{1}{\sqrt[N]{P(w_1, w_2, \ldots, w_N)}}$$

$$Perplexity = \frac{1}{\sqrt[N]{\prod_{i=1}^{N} p(w_i)}}$$

After taking log and simplifying we get:

$$\text{Perplexity} = e^{-\sum_{i=1}^{N} \log_e(P(w_i))/N}$$

2. **Bleu Score**

Given a machine translation it tells how good is the machine translation.It indicates how similar the candidate text is to the reference texts that are provided in the dev or test set.

BLEU Score	Interpretation
< 10	Almost useless
10 - 19	Hard to get the gist
20 - 29	The gist is clear, but has significant grammatical errors
30 - 40	Understandable to good translations
40 - 50	High quality translations
50 - 60	Very high quality, adequate, and fluent translations
> 60	Quality often better than human

$$\text{BLEU} = \underbrace{\min\Big(1, \exp\big(1 - \frac{\text{reference-length}}{\text{output-length}}\big)\Big)}_{\text{brevity penalty}} \underbrace{\Big(\prod_{i=1}^{4} precision_i\Big)^{1/4}}_{\text{n-gram overlap}}$$

with

$$precision_i = \frac{\sum_{\text{snt} \in \text{Cand-Corpus}} \sum_{i \in \text{snt}} \min(m^i_{cand}, m^i_{ref})}{w^i_t = \sum_{\text{snt'} \in \text{Cand-Corpus}} \sum_{i' \in \text{snt'}} m^{i'}_{cand}}$$

where

- m^i_{cand} is the count of i-gram in candidate matching the reference translation
- m^i_{ref} is the count of i-gram in the reference translation
- w^i_t is the total number of i-grams in candidate translation

- **Brevity Penalty:**

It penalises very short candidates as almost all the words in candidate can be present in reference if the candidate is very short compared to reference.

- **Disadvantages of Bleu Score:**
  - Not good at capturing meaning and grammar of a sentence. does not consider long range dependencies.
  - No difference between content and function words
  - The BLEU metric performs badly when used to evaluate individual sentences as it is a corpus based metric.
  - It does not take into consideration order of the n-grams in the reference and candidate sentence

Why does Pruning weights below a threshold work?

During pruning we prune weights that are redundant for our network. Weights with less magnitude are redundant because they have a smaller impact on the output of the network. This is because neural networks use a non-linear activation function, such as the sigmoid or ReLU function, after each layer of weights. These

activation functions squash the input values into a smaller range, which has the effect of attenuating the impact of small weights.

Three types of pruning schemes we implemented:

- **Class Blind:**
  - Sort the parameters of whole model. Find the threshold for pruning such that x% of weights is pruned.

attachment:de0042c1-58e2-4daa-843d-e159dc6b7184:ExampleScene.mp4

- Sort the parameters of each class and within each class find the threshold for pruning such that x% of weights in all classes of weights are pruned.

attachment:44a5c9d9-26c1-4f31-a0fb-9b79c3388608:class-uniform.mp4

- **Class Distribution:**
  - For each class c, weights with magnitude less than $\lambda\sigma_c$ are pruned. Here, σ_c is the standard deviation of that class and λ is a universal parameter chosen such that in total, x% of all parameters are pruned.

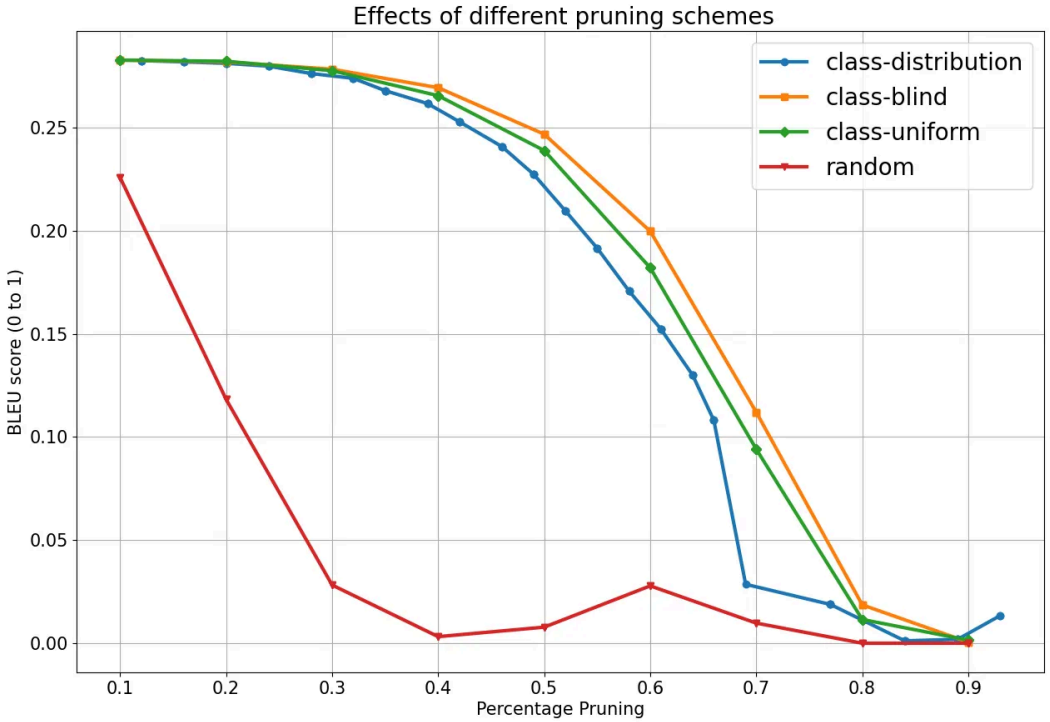
Summary of Strategies

Strategy	Layer Awareness	Balanced Across Layers	Adaptiveness	Tuning Required
Class-Blind	❌	❌	❌	Low
Class-Uniform	✅ (equal treatment)	✅	❌	Medium

Class-Distribution	✅ (statistically)	Adaptive	✅	High

Comparison of these three pruning schemes:

We can see that class-blind performs better than both class-uniform and class-distribution.

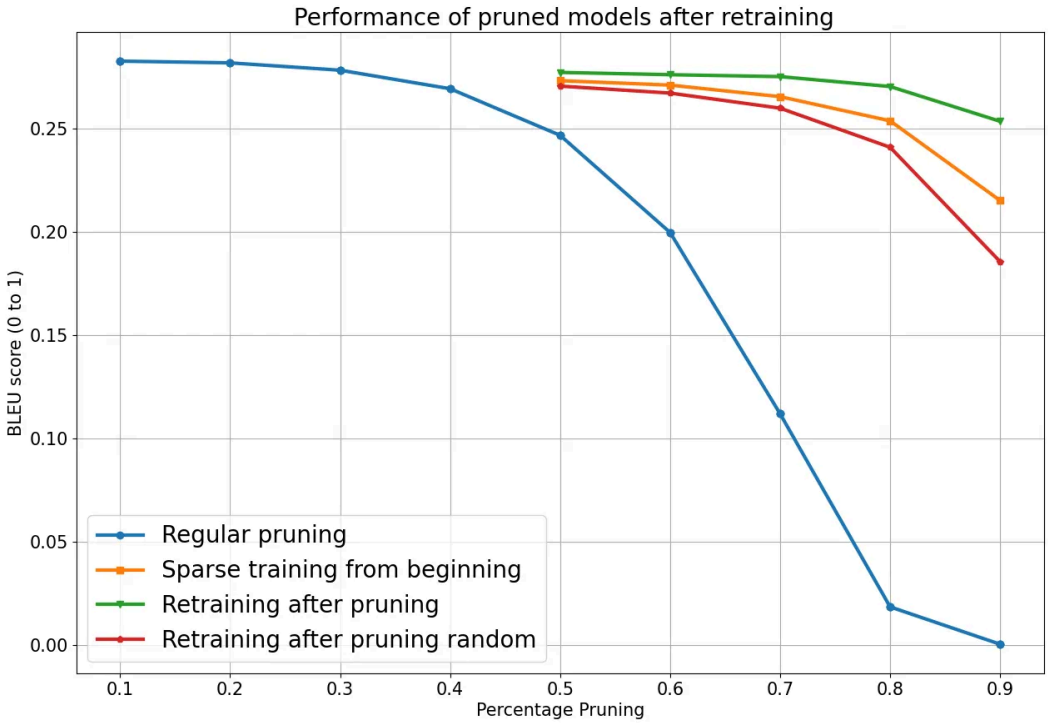


- **Random Global Pruning -** x% of weights are randomly pruned.

Magnitude pruning works better than random pruning, making it clear that weights with less magnitude are more likely to be redundant. All 3 kinds of magnitude pruning outperform random pruning.

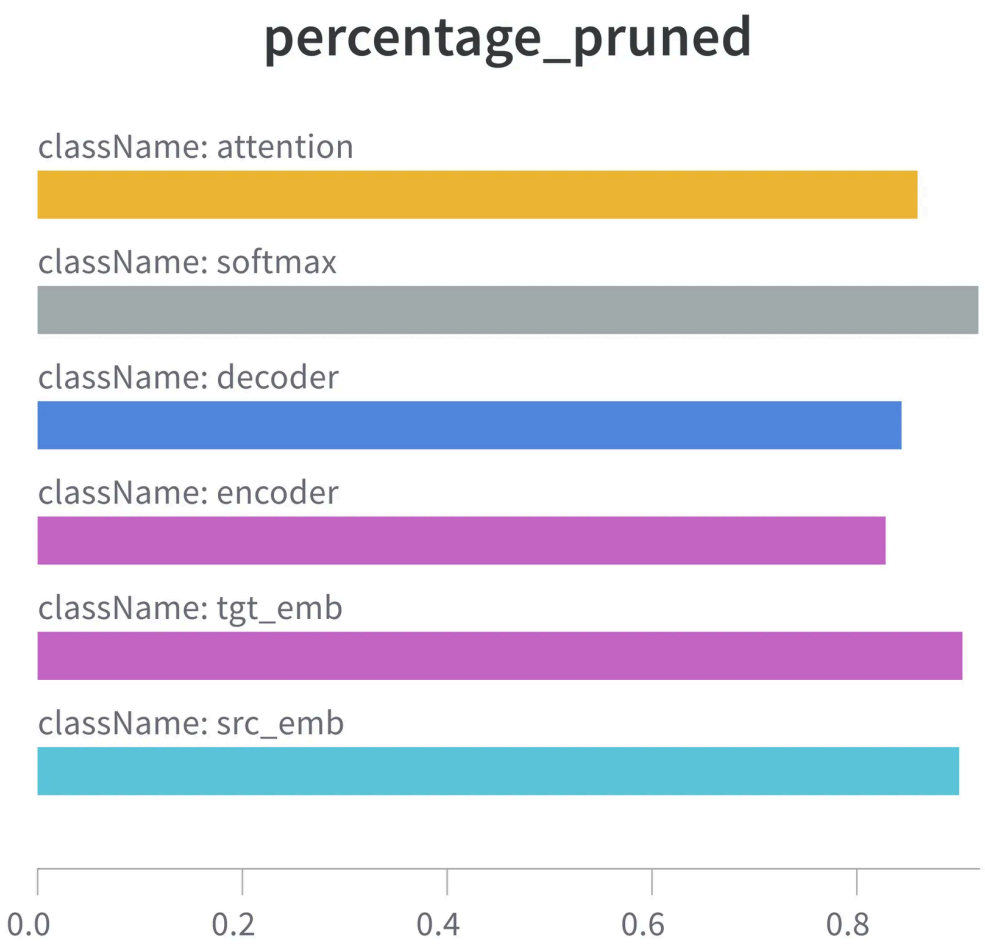
Comparison Of Retraining And Sparse From Begin Models:

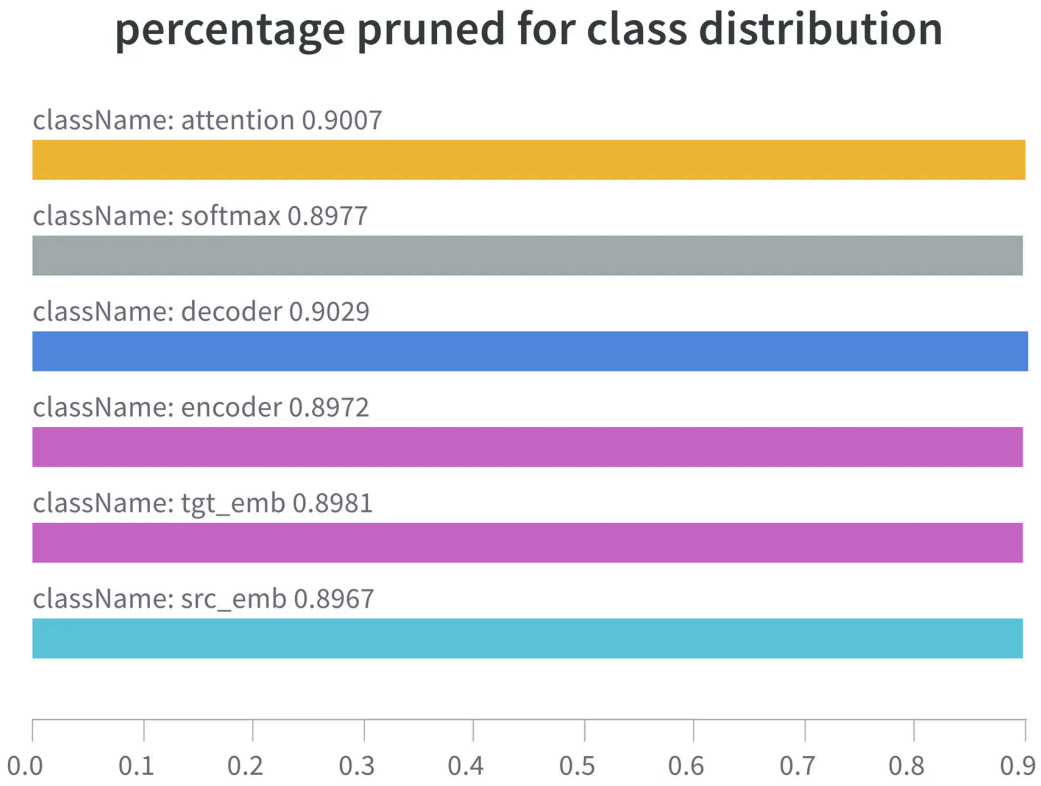
Comparison of random pruning coupled with retraining is included.



Graph shows that retraining is better, and it makes even random pruning much better. We can also use the sparse structure obtained during pruning to train sparse model from scratch. As we can see it is worse than retraining.

Percentage Pruned Per Class of Weights in Class-blind and Class-distribution for 90 percent pruning:

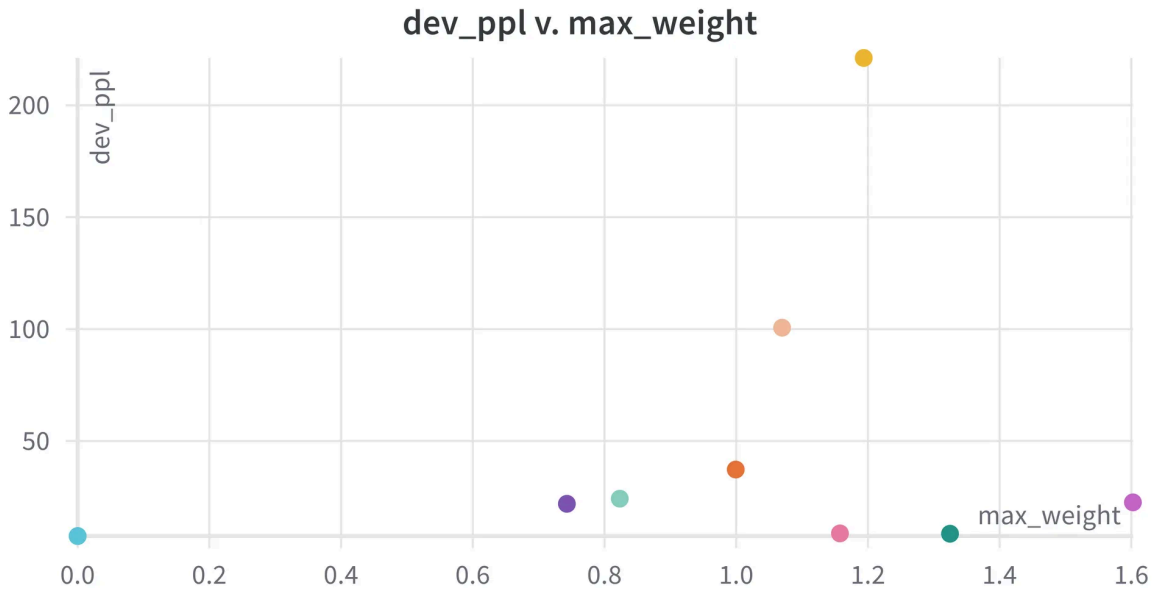

percentage_pruned
className: attention
className: softmax
className: decoder
className: encoder
className: tgt_emb
className: src_emb
0.0
0.2
0.4
0.6
0.8



Perplexity Change with pruning of each class of weights:

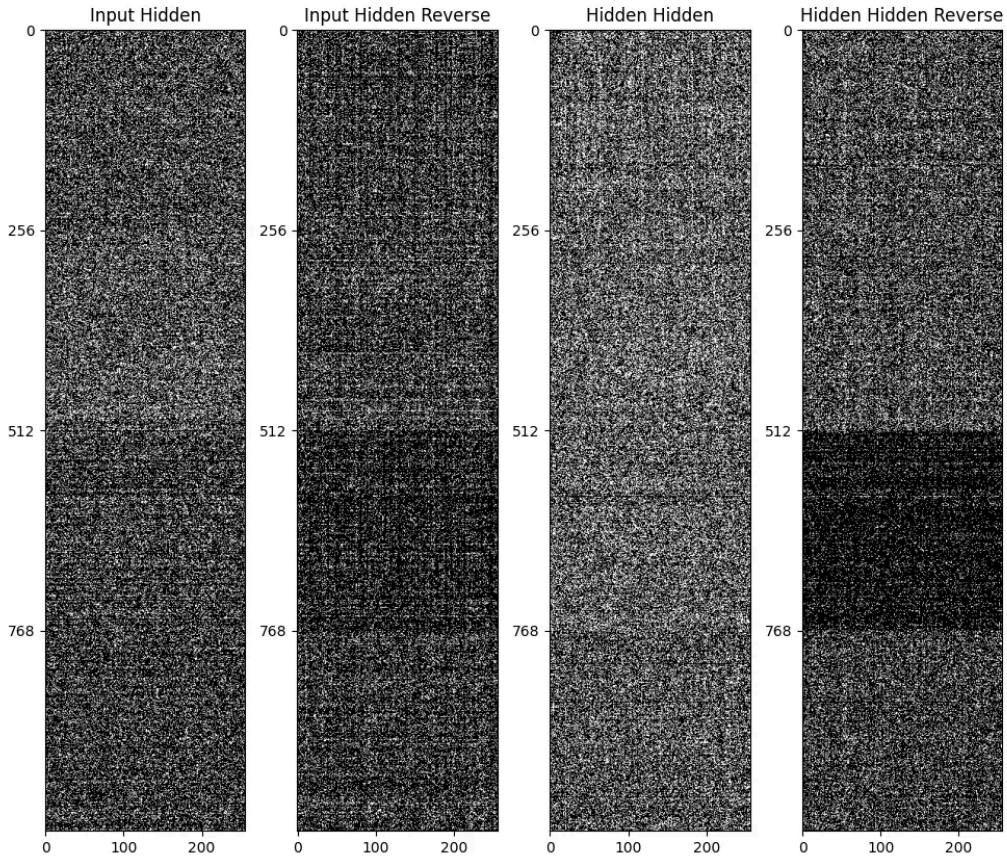
The graph shows that that some layers give higher perplexity change even though same number is pruned. This means those layers are more important and pruning them equally is not a good strategy. Interestingly last layers (softmax and attention) are more important than initial (target and source embeddings).

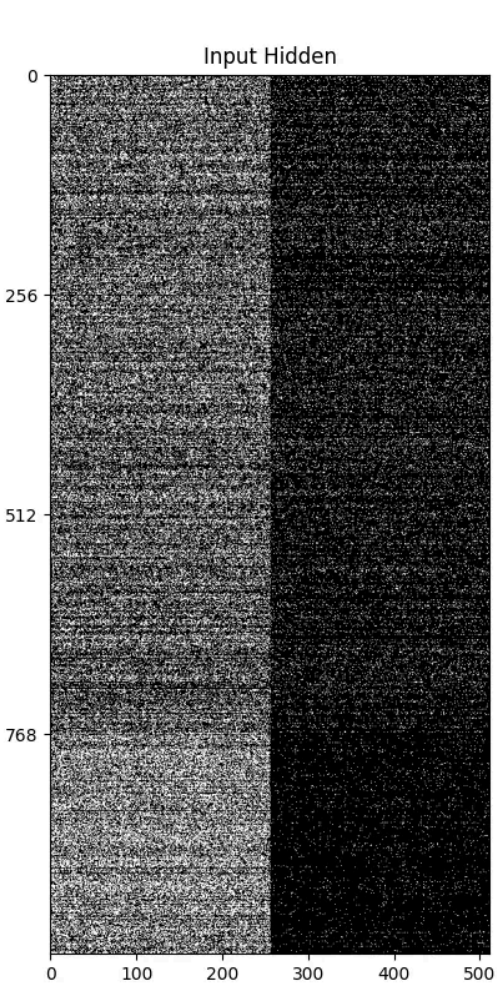
Change in perplexity vs max weight removed of that class

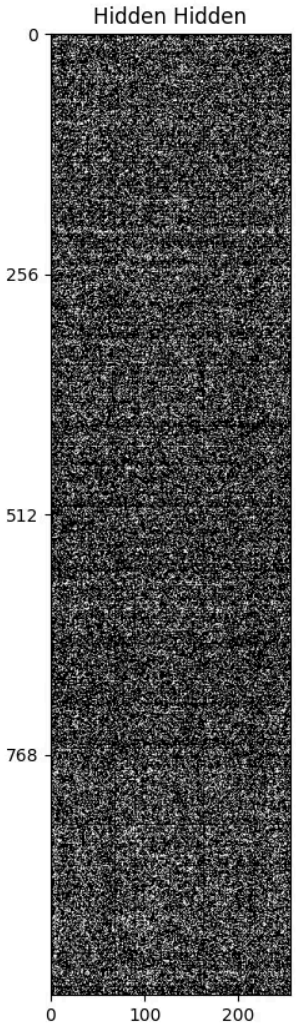


Encoder and Decoder Weights Visualisation

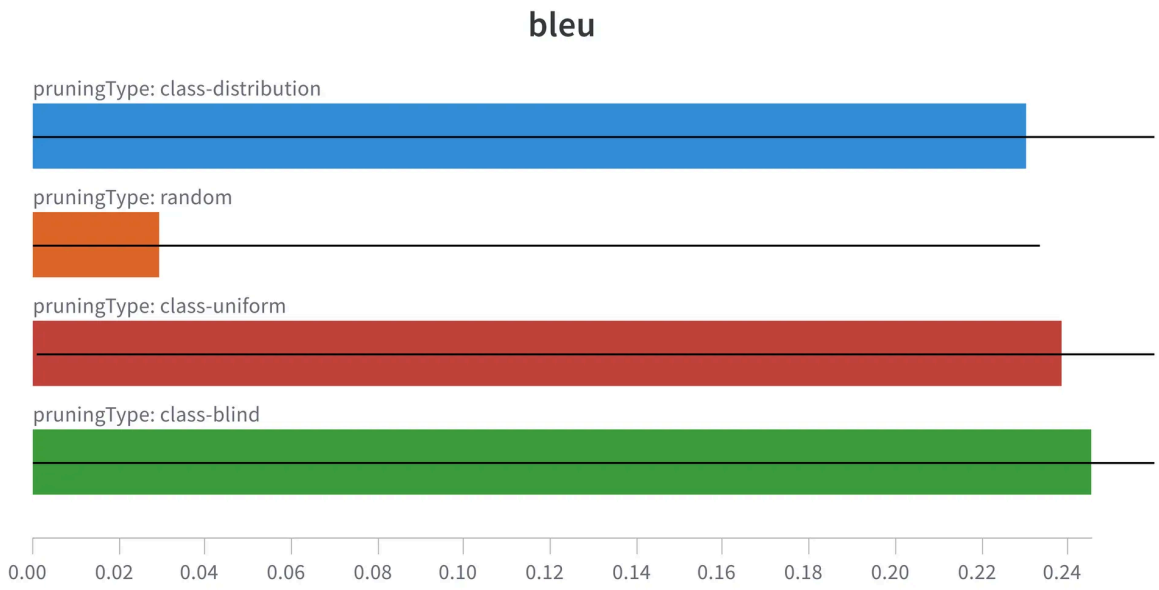
a) Encoder:


Input Hidden
Input Hidden Reverse
Hidden Hidden
Hidden Hidden Reverse
0
256
512
768
0
100
200





Median Bleu Score Comparison:



Future Plans

- **Analyze Current Results:** The results obtained for Transformer model so far are not as expected. We plan to investigate and understand the reasons behind this, and make necessary adjustments to our approach.
- **Random Pruning and Retraining for Transformer Model:** So far, random pruning and retraining have been done for the LSTM model. We aim to extend this by performing random pruning and subsequent retraining for the Transformer model as well.
- **Pruning for BERT-infused NMT Model:** We plan to experiment with pruning on a BERT-infused Neural Machine Translation (NMT) model to observe if introducing pretrained contextual embeddings affects pruning behavior and model robustness.
- **Explore Different Pruning Techniques:** In addition to magnitude-based unstructured pruning, we will include and experiment with other pruning techniques like Optimal Brain Damage (OBD) and SNIP to compare their effects on model performance.

- **Block Pruning for Transformer (if time permits):** If time allows, we intend to try block pruning methods for the Transformer model as suggested in the paper Pruning Neural Networks with Block Sparsity.