Mixture of Experts

SAiDL 2024 Assignment

Sasmit Datta

1 Introduction

Mixture of Experts (MoE) is a machine learning framework designed to integrate multiple models or "experts" which handle different tasks in a parallel fashion for improved prediction performance. In the context of this report the MoE layer is an assortment of MLPs.

I developed two baseline models: one consisting of stacked two LSTMs with an intermediate MLP "expert" layer, and another one with an MoE layer.

My primary focus was on exploring various gating strategies for these networks, including **top-k gating**, **top-k mask**, **noisy top-k**, and the incorporation of **importance** and **load** losses, to optimize the allocation of tokens among experts. I also plotted router distributions for these different strategies.

2 Mixture of Experts

The mathematical formulation of the MoE model is given by:

$$\mathbf{y} = \sum_{i=1}^{N} G_i(\mathbf{x}) \cdot E_i(\mathbf{x}) \tag{1}$$

where $\mathbf{y}$ is the output, $\mathbf{x}$ is the vector input, N is the number of experts, $G_i(\mathbf{x})$ is the gating function output for the i-th expert, determining its contribution weight, and $E_i(\mathbf{x})$ is the output of the i-th expert for input $\mathbf{x}$.

2.1 Architecture of Expert

For models of both datasets, the expert is a simple MLP with one **hidden layer** of **128 dimensions**. A **ReLU activation** is used before the input enters the hidden layer and **dropout** with $\boldsymbol{p = 0.5}$ is used at the end.

3 Model Variations

For each dataset I trained six different models, a baseline LSTM, and 6 LSTM's with an intermediate MoE layer with different gating strategies and losses. The models with MoE have same architecture for each dataset described in their respective sections. The gating strategies and losses are explained in Section 6.

Model Name	**About**
LSTM Baseline	Two LSTMs with intermediate expert layer
MoE Naive	Two LSTMs with MoE layer with Naive Top-K Gating
MoE Mask	Two LSTMs with MoE layer with Top-K Mask Gating
MoE Noisy TopK	Two LSTMs with MoE layer with Noisy Top-K Gating
MoE Importance	Two LSTMs with MoE layer with Top-K Mask and $L_{\text{importance}}$
MoE Load	Two LSTMs with Moe layer with Noisy Top-K Gating and L_{Load}

Table 1: Model Performance on CoNLL dataset

4 CoNLL-2003 Dataset

The CoNLL-2003 dataset is a collection of annotated texts that is widely used for training and evaluating machine learning models on named entity recognition (NER) tasks.

It includes a set of tags for named entity recognition that denote the beginning (B) and inside (I) of named entities, as well as tags for non-entity segments (O). The entity types covered include persons (PER), organizations (ORG), locations (LOC), and miscellaneous entities (MISC), each of which is prefixed by either B- or I- to indicate the position of the token within the entity.

4.1 Preprocessing

For preprocessing the CoNLL-2003 dataset, I created a unified vocabulary from all tokens, reduced it to match **GloVe 6B 50-dimensional** embeddings. This process involved removing non-GloVe tokens, adding special **padding** and **unknown** tokens, resulting in a pruned vocabulary. Token-to-index mappings were established, followed by the creation of an embedding matrix populated with GloVe vectors for each token. A **vector of zeros** was used for **padding** token and while the **unknown** token had the default **GloVe** <unk> token assigned to it.

4.2 Pipeline

The tokenizer maps the words in a given data point to their corresponding indexes and post-pads the given sequence to a **maximum length** of **128** with 0's (index of padding token). The **NER tags** are already in their numerical form and hence just **padded with -100** for the cross entropy loss to ignore them.

4.3 Training

For the base model, I used an expert layer sandwiched between two **bidirectional LSTM** layers with **hidden dimension 128** and a linear layer as a head to map the outputs to **9 dimensions** (number of NER tags).

For the MoE models, I just replace the single expert layer with a mixture of experts layer with **8 experts** and $\boldsymbol{k = 2}$. I use a **batch size** of **32** and train all the models for **10 epochs** each.

4.4 Results

Model Name	**Validation Accuracy (%)**	**Validation Loss**
LSTM Baseline	97.08	0.1179
MoE Naive	97.08	0.1142
MoE Mask	97.26	0.1173
MoE Noisy TopK	97.31	0.1087
MoE Importance	97.40	0.1095
MoE Load	97.30	0.1133

Table 2: Model Performance on CoNLL dataset

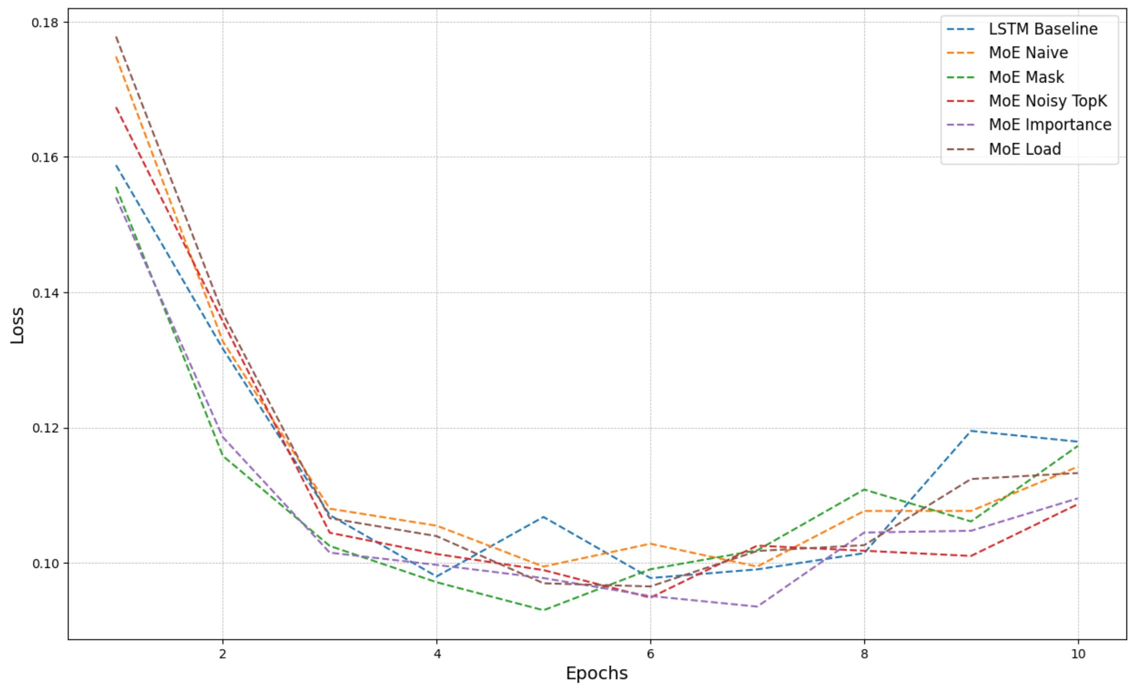


Figure 1: Loss Plots CoNLL

5 SQuAD 1.1 Dataset

The SQuAD 1.1 dataset is a collection of over 100,000 questions derived from approximately 500 Wikipedia articles, designed for training and evaluating question answering models.

Each entry in the dataset features a context (a text passage from an article), a question generated by human annotators based on the context, an answer to the question, and the starting position of the answer within the context. This structure supports extractive question answering, where models are tasked with identifying the exact span of text that answers the question.

5.1 Preprocessing

The preprocessing of the SQuAD 1.1 dataset involved **transforming** raw JSON data into structured **pandas DataFrames** by extracting contexts, questions, answers, and answer start positions.

Texts were tokenized using regex, maintaining punctuation as distinct tokens. Entries exceeding predefined **context** and **question length** of **256** tokens were filtered out. **Character indices** for **answer starts** were **converted into token indices** to align with the tokenized contexts.

A vocabulary was compiled from the tokenized texts, pruned to match **GloVe 6B 50d embeddings**, and **padding** and **unknown** tokens.

5.2 Pipeline

Like CoNLL, the tokenizer maps the words of a given context and question to their corresponding indexes and post-pads the two separate sequences with 0's to **maximum length** of **256**. Then with the help of the start index and the length of answer text, the end index of the answer is calculated.

5.3 Training

Like CoNLL, for the base model, I use an expert layer sandwiched between two **bidirectional LSTM** layers with **hidden dimension 128**. Unlike the CoNLL model, I use a head which maps the outputs to a **single dimension** as the model has to predict the position in the context where answer starts and ends.

I **concatenate** the **context and question** before passing it through the embedding layer, making the total input sequence length to 512. I **split the output** to get the **answer start** and the **answer end logits**. Essentially, I convert this into a classification task with number of classes equal to the sequence length of the context.

For the MoE models, I just replace the single expert layer with a mixture of experts layer with **8 experts** and $\boldsymbol{k} = \mathbf{2}$. I use a **batch size** of **32** and train all the models for **5 epochs** each.

5.4 Results

Model Name	Validation Accuracy (%)	Validation Loss
LSTM Baseline	11.45	8.770
MoE Naive	11.62	8.710
MoE Mask	11.47	8.766
MoE Noisy TopK	11.40	8.717
MoE Importance	11.46	8.794
MoE Load	11.76	8.710

Table 3: Model Performance on SQuAD dataset

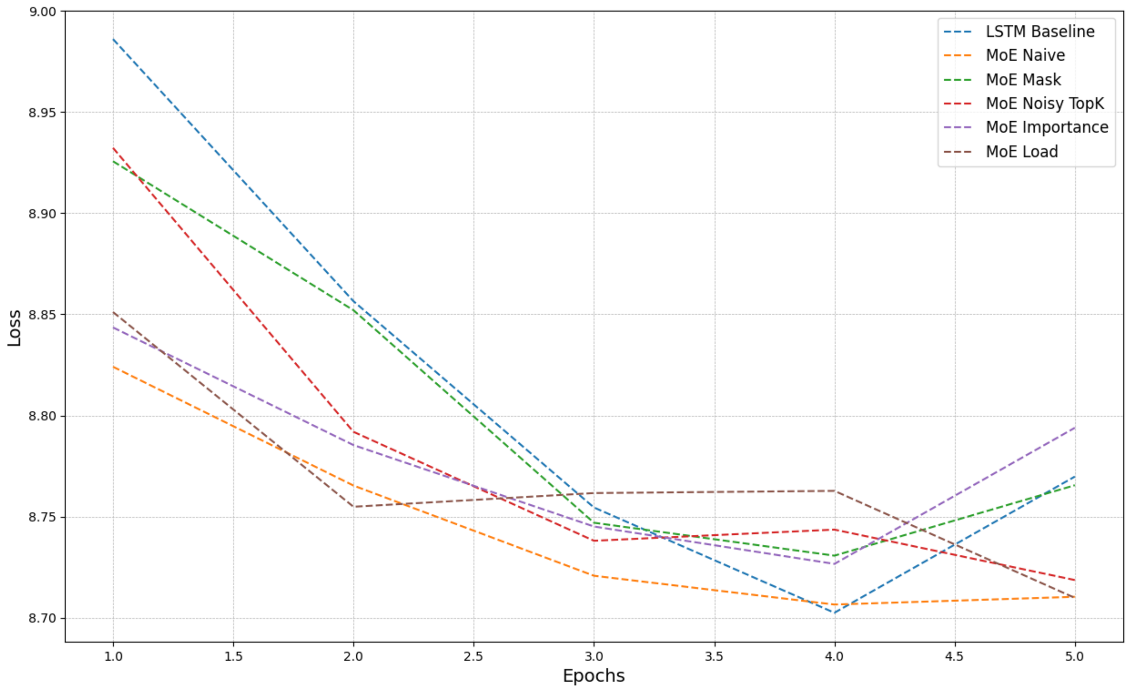


Figure 2: Loss Plots SQuAD

6 Gating Strategies and Losses

6.1 Naive Top-K Gating

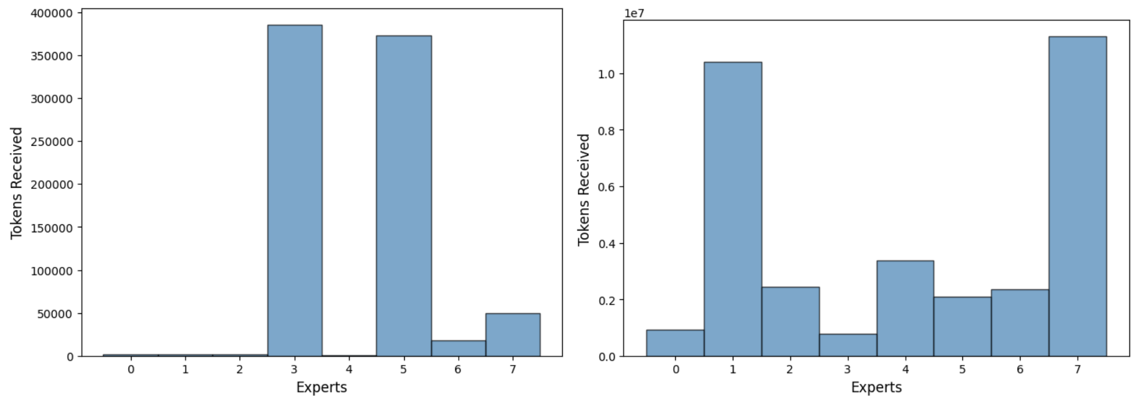
$$G(\mathbf{x}) = \text{Softmax}(\mathbf{x} \cdot \mathbf{W}_g) \odot \text{TopK}(\mathbf{x}, k) \tag{2}$$

where,

$$\text{TopK}(\mathbf{v}, k)_i = \begin{cases} 1 & \text{if } v_i \text{ is in the top } k \text{ elements of } v. \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

$\mathbf{W}_g$ is the trainable weight matrix, k is the number of experts per token and $\mathbf{x}$ is the input.

Essentially what this gating function does is naively multiplies the weights assigned by the router to the outputs of the top-k experts and adds them up.



(a) CoNLL-2003 (b) SQuAD 1.1

Figure 3: Naive Top-K Gating Expert Distributions

In Fig 3 we can see that the distribution of the model trained on the SQuAD dataset is more skewed. Since the model performs quite well on CoNLL dataset (Fig 3a) it might suggest that two experts are sufficient for the task at hand. Plus, model performance on SQuAD is terrible, hence the router might be trying its best to distribute tokens for more parameter usage.

6.2 Top-K Mask Gating

Instead naively multiplying the top-k weights above with the expert outputs, we first normalize them.

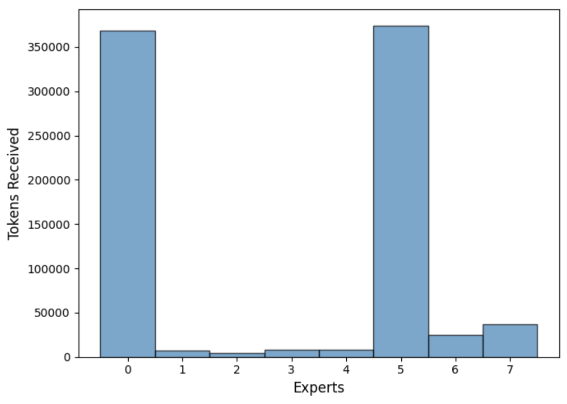
$$G_\sigma(\mathbf{x}) = \text{Softmax}(\mathbf{x} \cdot \mathbf{W}_g)\cdot$$

Therefore,

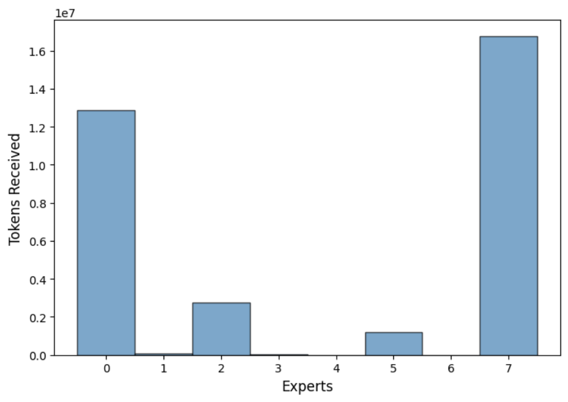
$$G(\mathbf{x})_i = \frac{G_\sigma(\mathbf{x})_i \text{TopK}(\mathbf{v}, k)_i}{\sum_{j=1}^{n} G_\sigma(\mathbf{x})_j \text{TopK}(\mathbf{v}, k)_j} \tag{4}$$

where n is the dimensions of our vector.

For the model being trained on CoNLL dataset, the router still continued to focus on just two experts but unlike before, the model trained on SQuAD only had its focus on four (Fig 4). Normalization causes the router to narrow in only certain experts, not utilising the model's full potential which is not ideal.



(a) CoNLL-2003



(b) SQuAD 1.1

Figure 4: Top-K Mask Gating Expert Distributions

6.3 Noisy Top-K Gating

Unlike standard Top-K gating, which deterministically selects the k experts with the highest gate values, noisy top-k adds randomness to the gating procedure, potentially allowing for exploration of less frequently chosen experts.

$$G(\mathbf{x}) = \text{Softmax}(\text{KeepTopK}(H(\mathbf{x}), k)) \tag{5}$$

where

$$H(\mathbf{x})_i = (\mathbf{x} \cdot \mathbf{W}_g)_i + X \cdot \text{Softplus}((\mathbf{x} \cdot \mathbf{W}_{\text{noise}})_i) \tag{6}$$

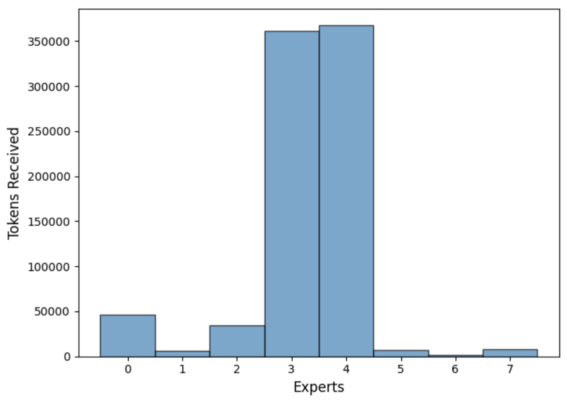
where $X \sim \mathcal{N}(0,1)$, $\mathbf{W}_{\text{noise}}$ is the trainable weight matrix which controls the amount of noise per expert,

$$\text{Softplus}(z) = \begin{cases} \log(1+e^z) & \text{if } z < 20 \\ z & \text{otherwise} \end{cases} \tag{7}$$

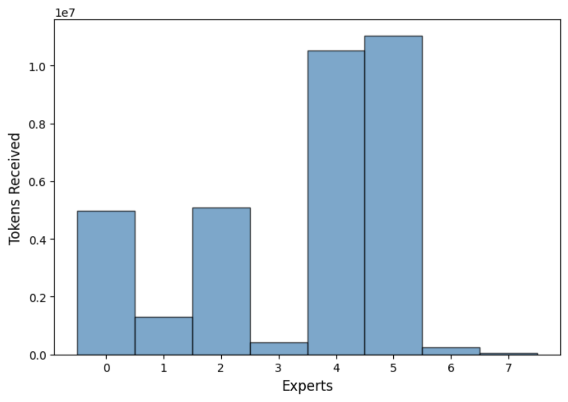
and

$$\text{KeepTopK}(v,k)_i = \begin{cases} v_i & \text{if } v_i \text{ is in the top } k \text{ elements of } v. \\ -\infty & \text{otherwise.} \end{cases} \tag{8}$$

$\mathbf{W}_g$ and $\mathbf{W}_{\text{noise}}$ were initialised with 0's for training.



(a) CoNLL-2003



(b) SQuAD 1.1

Figure 5: Noisy Top-K Gating Expert Distributions

The added stochasticity forced the router to use other experts. In spite of this, the model being trained on CoNLL still focused only on two experts (Fig 5a). This could allude to the router not heeding much to the noise element of $H(\mathbf{x})_i$.

6.4 Importance Loss

Importance of an expert relative to a batch of training examples to be the batch-wise sum of the gate values for that expert.

$$\text{Importance}(\mathbf{X}) = \sum_{\mathbf{x} \in \mathbf{X}} G(\mathbf{x}) \tag{9}$$

An additional loss $L_{\text{importance}}$ defined through this which encourages all the experts to have equal importance.

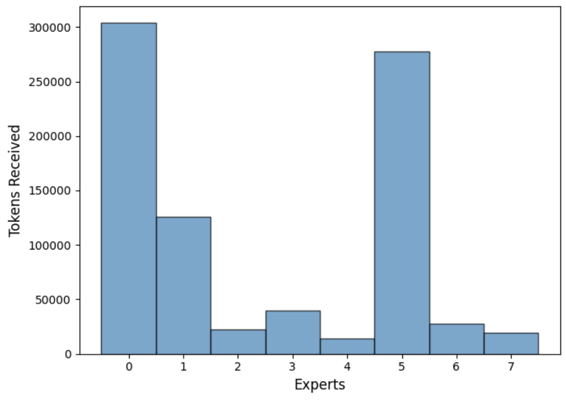
$$L_{\text{importance}}(\mathbf{X}) = w_{\text{importance}} \cdot CV(\text{Importance}(\mathbf{X}))^2 \tag{10}$$

where $w_{\text{importance}}$ a hand-tuned weighing factor and

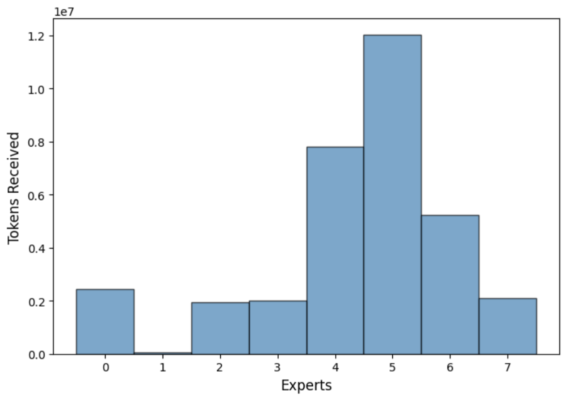
$$CV(\text{Importance}(\mathbf{X})) = \frac{\sigma(\text{Importance}(\mathbf{X}))}{\mu(\text{Importance}(\mathbf{X})) + \epsilon} \tag{11}$$

where ϵ is a small term added for numerical stability.

$\boldsymbol{w}_{\text{importance}} = \mathbf{0.1}$ was being used for training.



(a) CoNLL-2003



(b) SQuAD 1.1

Figure 6: Top-K Mask Gating with $L_{\text{importance}}$ Expert Distributions

Explicitly penalizing the narrowing of expert selection of router of the model trained on the CoNLL dataset finally forced it to select other experts for its tasks (Fig 6a). The SQuAD model still tried to decrease its loss by maximising use of most of the resources available to it (Fig 6b).

6.5 Load Loss

The load loss is a mechanism designed to ensure a balanced distribution of workload among the various experts in noisy top-k gating.

$$P(\mathbf{x}, i) = \Pr\left((\mathbf{x} \cdot \mathbf{W}_g)_i + X \cdot \text{Softplus}((\mathbf{x} \cdot \mathbf{W}_{\text{noise}})_i) > \text{kth\_excluding}(H(\mathbf{x}), k, i)\right) \tag{12}$$

where kth_excluding(v, k, i) means the k-th highest component of v, excluding component i and $X \sim \mathcal{N}(0, 1)$ is clearly a random variable. So the given probability can be evaluated to

$$P(\mathbf{x}, i) = \Phi\left(\frac{(\mathbf{x} \cdot \mathbf{W}_g)_i - \text{kth\_excluding}(H(\mathbf{x}), k, i)}{\text{Softplus}((\mathbf{x} \cdot \mathbf{W}_{\text{noise}})_i)}\right) \tag{13}$$

where Φ is the CDF of the standard normal distribution. Therefore we can define load as

$$\text{Load}(\mathbf{X})_i = \sum_{\mathbf{x} \in \mathbf{X}} P(\mathbf{x}, i) \tag{14}$$

Finally, load loss can be defined as

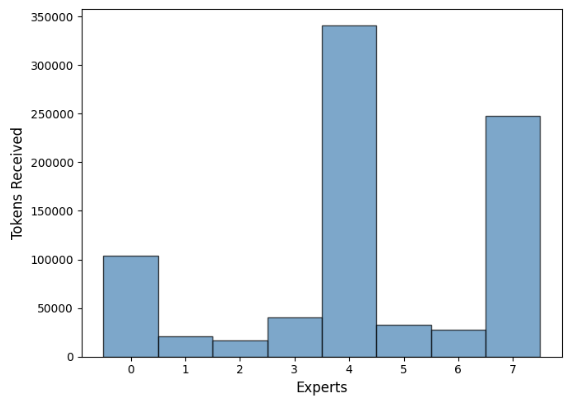
$$L_{\text{Load}}(\mathbf{X}) = w_{\text{load}} \cdot CV(\text{Load}(\mathbf{X}))^2 \tag{15}$$

where w_{load} a hand-tuned weighing factor and

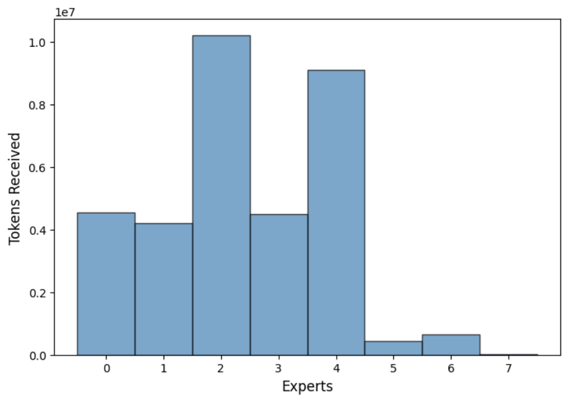
$$CV(\text{Load}(\mathbf{X})) = \frac{\sigma(\text{Load}(\mathbf{X}))}{\mu(\text{Load}(\mathbf{X})) + \epsilon} \tag{16}$$

where ϵ is a small term added for numerical stability.

$\boldsymbol{w}_{\text{load}} = \mathbf{0.1}$ was being used for training.



(a) CoNLL-2003



(b) SQuAD 1.1

Figure 7: Top-K Mask Gating with L_{load} Expert Distributions

The noisy top-k router of the CoNLL model finally budged after being forced to balance its loads (Fig 7a). The SQuAD model spreads the use of its experts with the load loss (Fig 7b) as compared to before.

7 Conclusion

With the gating strategies covered above, we can clearly see that the models being trained on **CoNLL didn't diverge from two experts** - most likely due to the fact that it **only need those two experts** to get good results on the dataset. On the other hand, the models being trained on **SQuAD** tried to **maximise the use of the resources** available to it since they were getting quite high log-likelihoods as loss (Fig 2).