Student: Megh Shah
Mentor: Dr. Tara Sainath, Google, Inc.
https://sites.google.com/site/tsainath/

SAIDL Induction Documentation:

Part 1: Softmax

Description of Model:

Description of what the Convolutional Neural Network is:

- The first block of code initializes the training parameters and gives a commencement to our CNN.
- The second block is used to define the various classes of data/images we need to classify data into.
- Then comes the definition of CNN. So basically, our CNN has two convolutional layers with dropout after each layer. The dropout after each layer prevents overfitting i.e. the over-optimisation of the CNN which makes it less accurate or confused in layman's words. Then we also have the dense layer.
- This is then followed by the main part of the training code which has the softmax function. A softmax is a probability function which helps us in classifying the data into the ten classes we have taken into account.(which is our primary objective over here). Softmaxes can be of various types depending on the distribution of data we possess.
- This is then followed by testing, validation and visualization of false predictions/outputs.

SOFTMAX TYPES USED IN THE FOLLOWING PROJECT:

1) CROSS ENTROPY:

The fundamental equation of CROSS ENTROPY SOFTMAX is shown as below:

$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

W stands for the weights of the CNN.
K is a variable that takes consecutive values.

Below listed are some of the major characteristics of CROSS ENTROPY softmax:

- The time complexity(O(N)) of this type of Softmax function is N as we have summation over the interval k in the denominator of the function as shown above.

- The softmax function takes a vector of real numbers, typically referred to as logits, and transforms them into a probability distribution. It squashes the logits into values between 0 and 1, while ensuring that the sum of all probabilities is equal to 1. The softmax function is defined as follows for a given class.

2) GUMBEL:

The equation of gumbel type softmax is listed below:

$$y_i = \exp((G_i + \log(\pi_i)) / \tau) / \Sigma_j \exp((G_j + \log(\pi_j)) / \tau)$$

The Gumbel distribution is a continuous probability distribution that can be used to generate samples that approximate discrete samples. It is defined by two parameters, α and β, where α represents the location and β controls the shape of the distribution. The Gumbel distribution is characterized by its heavy tail and has the following probability density function (PDF).

Below listed are some of the major characteristics of GUMBEL softmax:

- Similar to the cross entropy softmax representation, the gumbel softmax also portrays a time complexity(O(N)) of N.

3) ADAPTIVE:

Adaptive Softmax is a speedup technique for the computation of probability distributions over words. The adaptive softmax is inspired by the class-based hierarchical softmax, where the word classes are built to minimize the computation time. Adaptive softmax achieves efficiency by explicitly taking into account the computation time of matrix-multiplication on parallel systems and combining it with a few important observations, namely keeping a shortlist of frequent words in the root node and reducing the capacity of rare words.

In our case where we use a distribution where the object falling into each class does not vary too much i.e. we call it a uniformly distributed system in plain words, the adaptive softmax has limited advantages.

4) HIERARCHICAL:

Hierarchical Softmax is an alternative to softmax that is faster to evaluate: it is time to evaluate compared to for softmax. It utilizes a multi-layer binary tree, where the probability of a word is calculated through the product of probabilities on each edge on the path to that node.
The main motivation behind this methodology is the fact that we're evaluating about logarithm to base 2 of *V* instead of *V*:

$$O(V) \rightarrow O(\log_2 V)$$

which is a dramatic change in computational complexity and number of operations needed for the algorithm. We do it with the usage of the binary tree, where leaves represent probabilities of words; more specifically, leave with the index *j* is the *j-th* word probability and has position *j* in the output softmax vector.

Each of the words can be reached by a path from the root through the inner nodes, which represent probability mass along that way. Those values are produced by the

usage of simple sigmoid function as long as the path we're calculating is simply the product of those probability mass functions defined with:

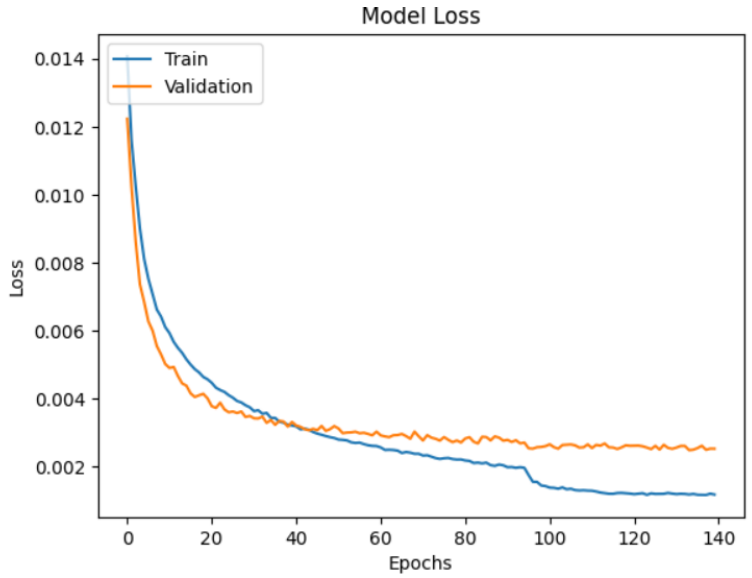
$$\sigma(x) = \frac{1}{1 + exp(-x)}$$

The above expression shows the fundamental function of Hierarchical Softmax.

Results

1) Cross entropy

- Shown below is the training curve for cross-entropy. We observe the dip in training loss after approximately 100 epochs, which indicates slight overfitting. The model is roughly converged after 100 epochs as the validation loss becomes saturated/stagnant after roughly 100 epochs while the training loss further subdues.



- We clearly notice that the loss decreases as the number of epochs increases and then saturates at around 0.0010(for training) and 0.0030(for validation) at around 100th epoch.

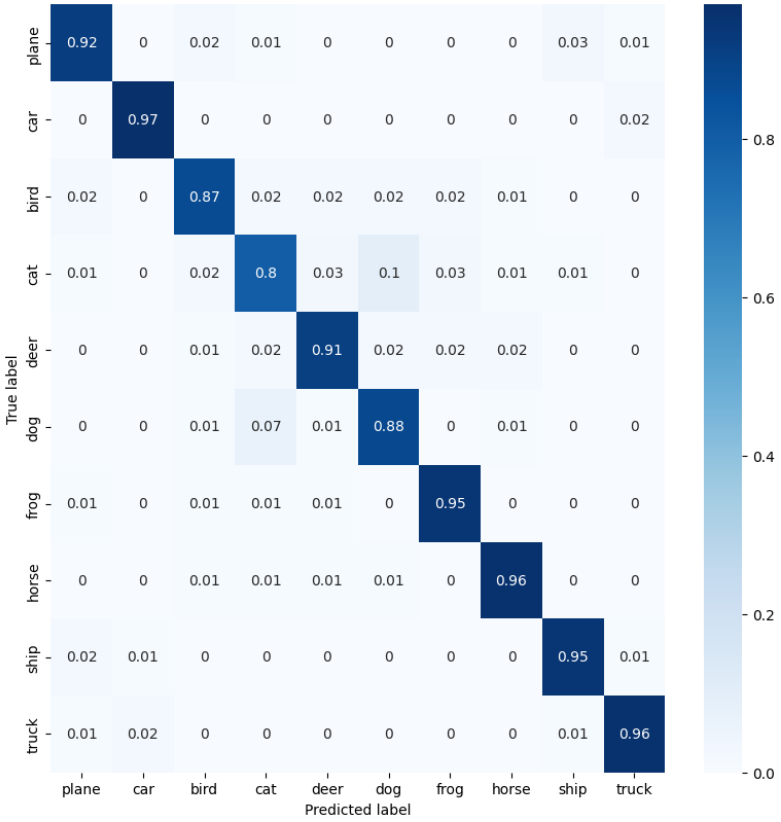
- The loss is around 0.015 at the first epoch and drops dramatically thereafter till epoch 20.

- Taking the trained cross-entropy checkpoint, We observe an accuracy of 91.54% when cross entropy softmax is executed over 10000 test images.

- We also plot the confusion matrix over here. A confusion matrix is basically a table which gives us an analysis on the wrongly predicted images. It shows which of the wrong classes is selected and how many times.

- We observe that ‘car’ is the most accurately predicted class followed by ‘horse’ and ‘truck’.

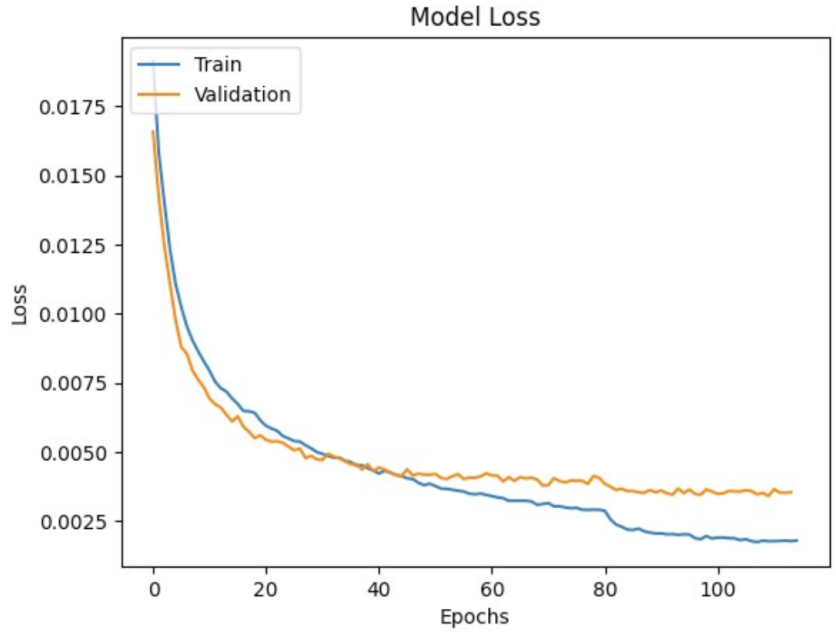
- The least accurate prediction is that of the ‘cat’ which is most confused with the ‘dog’ label, possibly because they resemble each other the most relative to the other labels. Similarly, the ‘dog’ label is mostly mistaken the most as the ‘cat’ label.



2) GUMBEL:

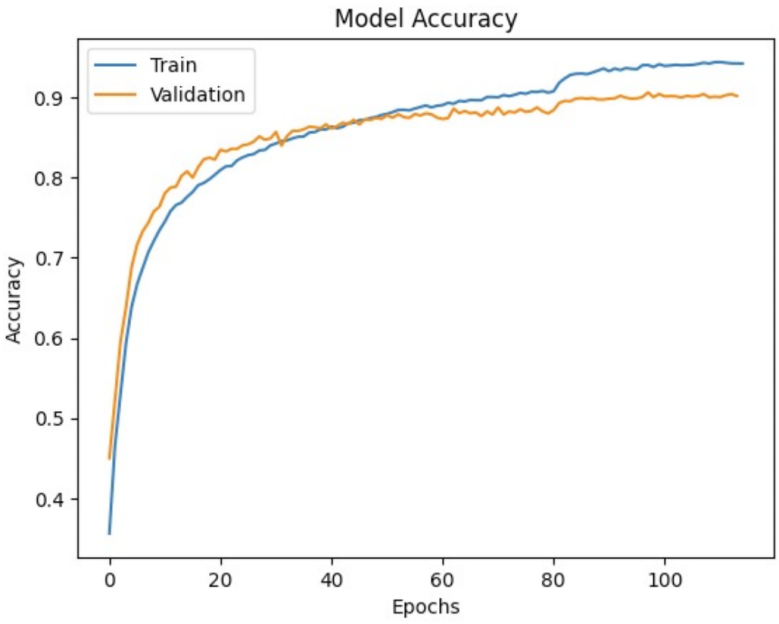
- We clearly notice that the loss decreases as the number of epochs increases and then saturates at around 0.0025(for training) and 0.0050(for validation) at around 80th epoch.

- The loss is around 0.02 at the first epoch and drops dramatically thereafter till epoch 20.



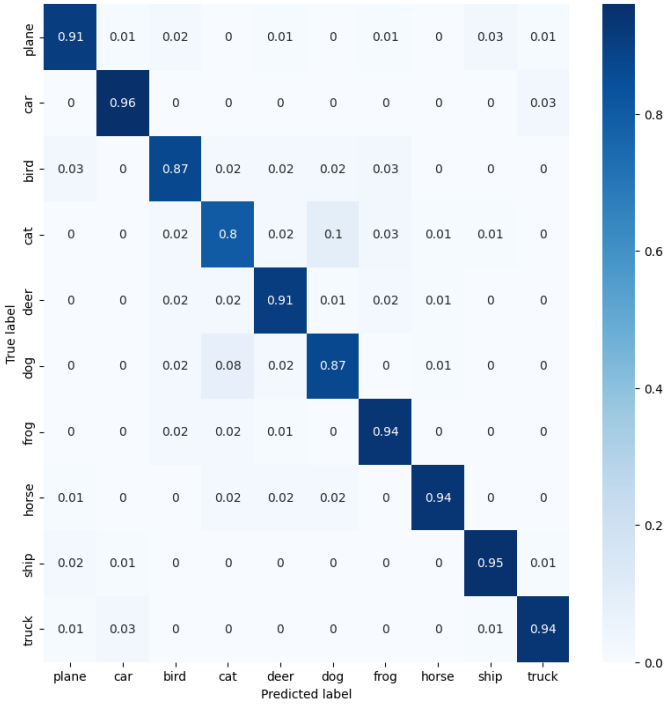
- We clearly notice that the accuracy shoots up as the number of epochs increases and then saturates at around 0.95(for training) and 0.90(for validation) at around 80th epoch.

- The accuracy is around 0.2 at the first epoch and shoots up drastically thereafter till epoch 20.



- We observe an accuracy of 91.02% when cross entropy softmax is executed over 10000 test images.

- We also plot the confusion matrix over here. A confusion matrix is basically a table which gives us an analysis on the wrongly predicted images. It shows which of the wrong classes is selected and how many times.



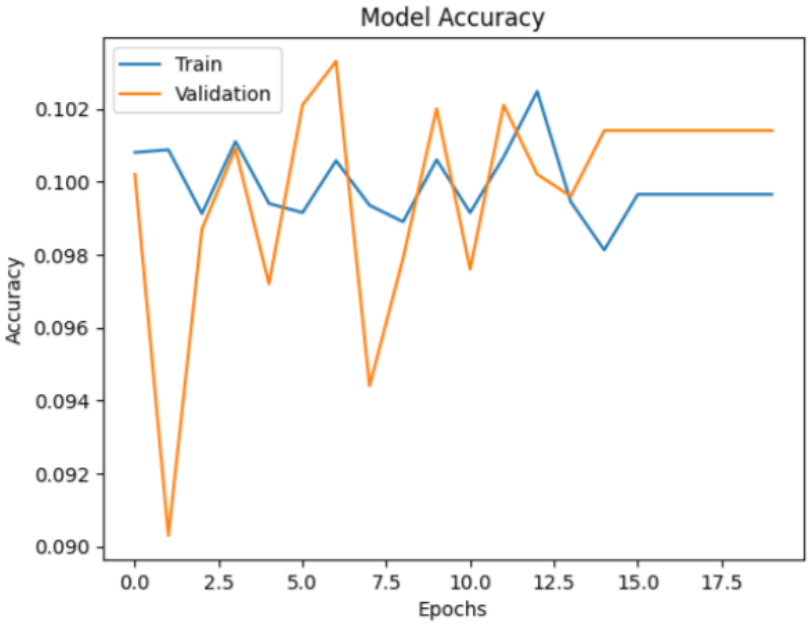
- We observe that ‘car’ is the most accurately predicted class(Accuracy=0.96) like the cross-entropy case followed by ‘ship’ and then ‘truck’ and ‘horse’.

- The least accurate prediction is that of the ‘cat’ which is most confused with the ‘dog’ label, possibly because they resemble each other the most relative to the other labels. Similarly, the ‘dog’ label is mostly mistaken the most as the ‘cat’ label. This is exactly the same as the cross-entropy softmax case.

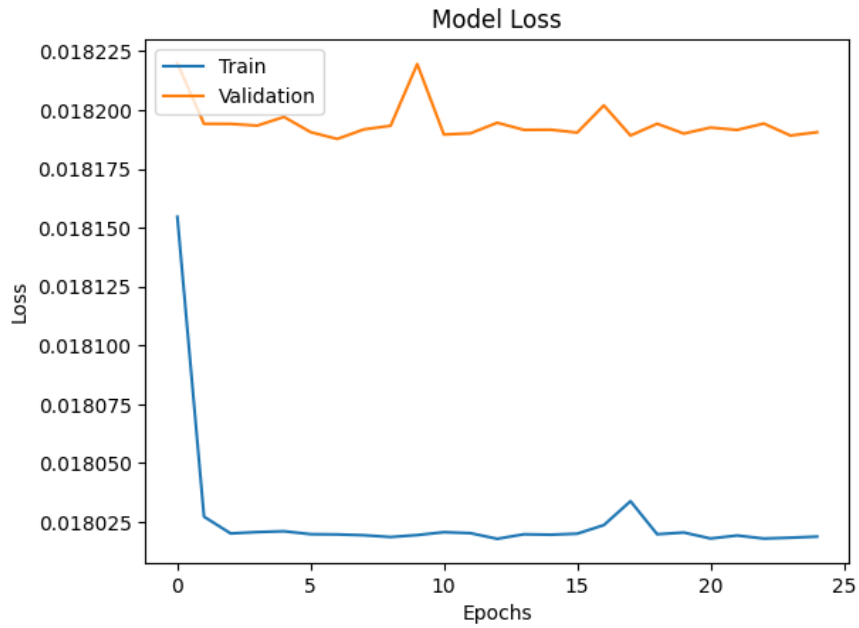
3) **HIERARCHICAL:**

We observe an accuracy of 10.00% when hierarchical softmax is executed over 10000 test images.

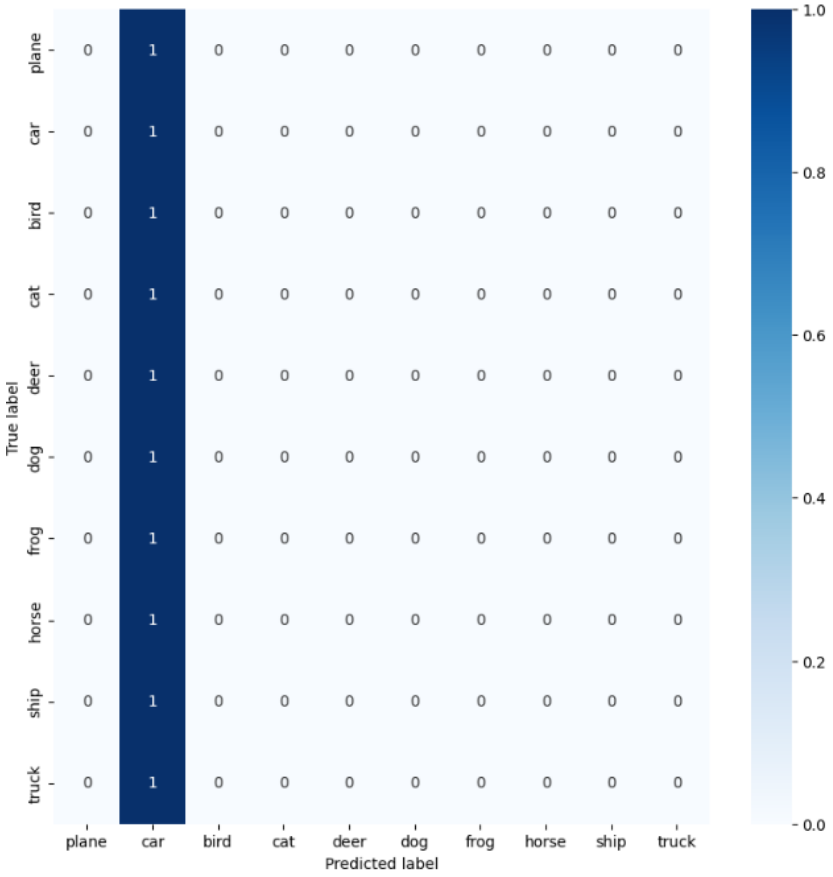
- Hierarchical softmax typically is used when we have a large number of classes which are unbalanced. Our CIFAR-10 dataset is exactly the opposite - a small number (10) of balanced classes.



- The accuracy clearly stabilizes near 0.10 as shown in the graph.



We also plot the confusion matrix over here. A confusion matrix is basically a table which gives us an analysis on the wrongly predicted images. It shows which of the wrong classes is selected and how many times.

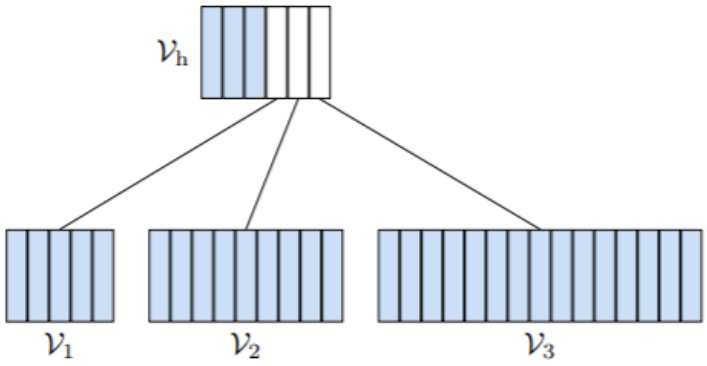


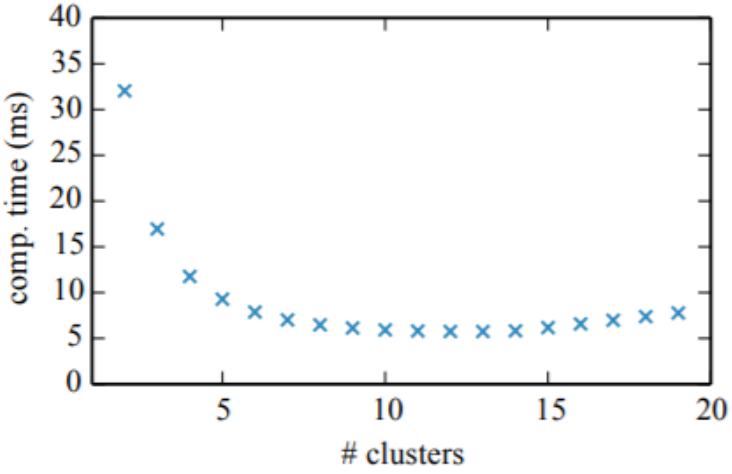
- The model predicts ‘car’ for all the test images.

- Hence the accuracy is 100% for the ‘car’ label and 0 for all other labels.

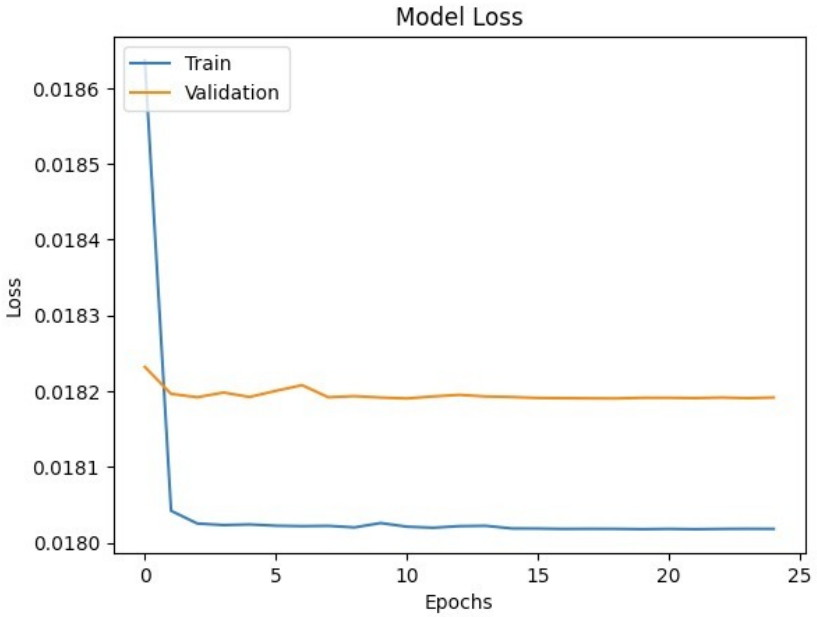
4) ADAPTIVE:

Adaptive Softmax is a speedup technique for the computation of probability distributions over words. The adaptive softmax is inspired by the class-based hierarchical softmax, where the word classes are built to minimize the computation time. Adaptive softmax achieves efficiency by explicitly taking into account the computation time of matrix-multiplication on parallel systems and combining it with a few important observations, namely keeping a shortlist of frequent words in the root node and reducing the capacity of rare words.





- We observe a poor model loss and accuracy of 10.04% when adaptive softmax is executed over 10000 test images. Adaptive softmax typically is used when we have a large number of classes which are unbalanced. Our CIFAR-10 dataset is exactly the opposite - a small number (10) of balanced classes.



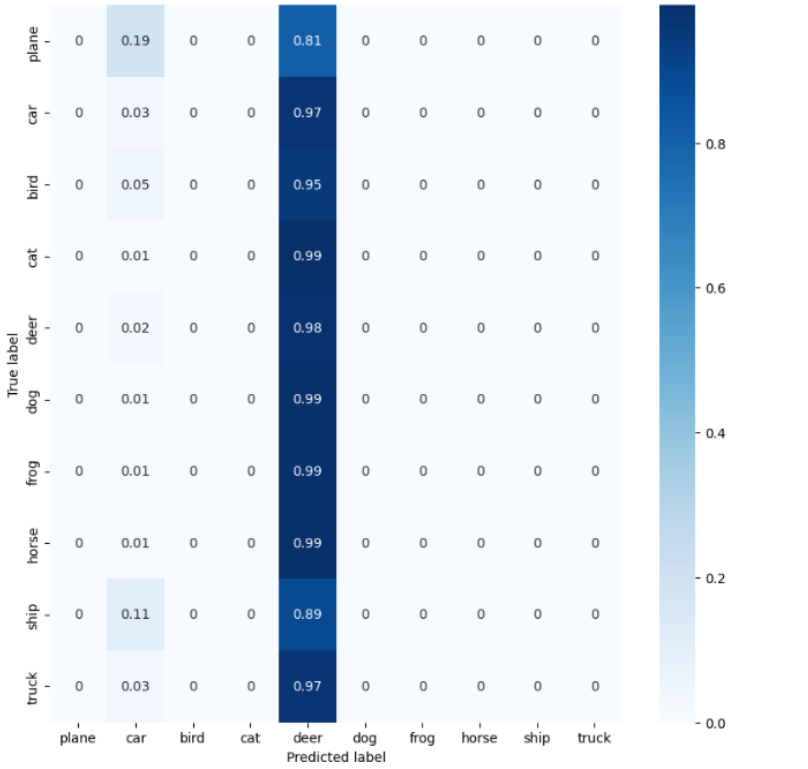
- This type of softmax first classifies the set of labels into classes based on the frequency of occurrence from highest to lowest.

- Therefore, an uneven distribution with some labels repeating very frequently to some coming up very seldom is the most suitable for the adaptive type of softmax.

- Our model is mainly an even distribution wherein each label has nearly similar probability of occurrence. Hence, this type of softmax is not the best one to use in our case.

- Hence, the results we get are not satisfactory.

We also plot the confusion matrix over here. A confusion matrix is basically a table which gives us an analysis on the wrongly predicted images. It shows which of the wrong classes is selected and how many times.



- This model has relatively very low accuracy.
- Most of the time, the model has predicted the ‘deer’ irrespective of what the actual picture suggests.
- The best accuracy is held by the ‘deer’ label which has an accuracy of 98%. 2% of the times the model has wrongly predicted ‘deer’ as a ‘car’.
- The second best accuracy can be seen for the ‘plane’ label. The model has predicted it correctly 19% of the time and has predicted ‘deer’ 81% of the time.
- The third best accuracy is for the ‘ship’ label i.e. 11%.
- The worst accuracy position is jointly held by the ‘cat’, ‘dog’, ‘frog’ and ‘horse’.

References:

- https://en.wikipedia.org/wiki/Softmax_function
- https://towardsdatascience.com/what-is-gumbel-softmax-7f6d9cdcb90e
- https://paperswithcode.com/method/gumbel-softmax#:~:text=Gumbel%2DSoftmax%20is%20a%20continuous,Categorical%20Reparameterization%20with%20Gumbel%2DSoftmax
- https://machinelearningmastery.com/cross-entropy-for-machine-learning/
- https://paperswithcode.com/method/adaptive-softmax
- https://github.com/facebookresearch/adaptive-softmax
- https://towardsdatascience.com/hierarchical-softmax-and-negative-sampling-short-notes-worth-telling-2672010dbe08
- https://paperswithcode.com/method/hierarchical-softmax#:~:text=Hierarchical%20Softmax%20is%20a%20is,the%20path%20to%20that%20node.

2. Part 2:

Basic Overview:

- We aim to create a model which can classify various sentences to HATE SPEECH/NOT HATE SPEECH.
- For this particular model, we are making use of the HASOC dataset.
- At a high level, we are translating a part of an English sentence into a Hindi(for various CMI values). CMI: The fraction of the sentence that has to be translated to Hindi. Here the CMI takes the following values: 0.2, 0.25, 0.5, 0.75 and 1.00.
- We train it using BERT(Bidirectional Encoder Representations for Transformers).
- We need to convert our dataset into a format on which BERT can be trained on.
- For that purpose we use something known as Tokenizing.
- After we train the model, we move forward to test it.
- We tested the model for **five values** of CMI. We also used two models of BERT.
- The table below lists the efficiency of the model for each value of CMI for each of the BERT models namely ‘bert-base-multilingual-cased’ and ‘’.

Explanation of the code:

- In the very first part of the code, we install google translate(install googletrans) and transformer.
- I also mounted(linked) my Google Drive to Colab in order to import some data files required.
- Then we import a set of packages which are necessary for the completion of our task.
- In the next step, we read our original data file (`english_2021.csv`) which has to be translated.
- After this is done, we actually translate the file according to the set CMI and save the file(`english_2021_translated_1p00.csv`).
- We have taken the translation of the sentence in the **Latin** script instead of the original **Devanagiri** script used in writing Hindi. This is done because the model can only train on Latin script and not on the typical Devanagiri one.
- The word to word translations have been taken for all the sentences. Not all sentences translated make complete sense as the sentence structure has not been considered for simplicity.

  For example, lets take the sentence “Good morning how was your day?” Code-mixing this becomes “Su prabhat how was your din?”

- Now we will use the translated data for training,validation and test purposes.
- The next block of code divides the whole dataset into training data(80%), validation data(10%) and test data(10%).

```
Number of training sentences: 3,843
Training: 3074
Validation: 384
Test: 385
```

- Then the tokenizing takes(explained above) place. Each word is expressed with a unique value which is then multiplied with a Single-Hot Vector.
- After this step, the actual training takes place(in multiple epochs).
- In the final few steps, training and validation plots are plotted.
- Finally, the testing takes place which gives us a resulting test efficiency for each value of CMI. The results are displayed in the table below.

Results:

	CMI	Accuracy(%)
BERT	0.00	79.80
	0.25	77.60
	0.50	76.40
	0.75	66.60
	1.00	66.10
mBERT	0.00	75.70
	0.25	76.00
	0.50	74.00
	0.75	64.90
	1.00	65.60
XLM-17	0.00	77.60
	0.25	78.60
	0.50	78.40
	0.75	66.80
	1.00	64.40

Trend of the Result:

- For both the cases mentioned above, we see that the efficiency increases initially with the CMI, reaches a maxima and then shows a steady decline as the CMI increases further.
- Case 1: The efficiency peaks at 79.80% when the CMI = 0.00 and then goes on decreasing to 66.10% as the CMI tends to 1.00.
  We see a visible drop in accuracy as the CMI goes up because it is harder for the model to work/predict on Hindi words as the model is mainly trained in the language English.
  Hence we see a decrease in the accuracy as the CMI increases i.e. number of Hindi words in the sentence increases.
- Case 2: Similar trend is visible for the second case as well. The efficiency slightly increases to 76.00% at CMI = 0.25 and then subsides to 65.60% at CMI = 1.00. We notice the same trend in mBERT as well because of the same reason as mentioned in the previous case i.e. the model has been mainly/predominantly been trained in English so it is harder for it to predict when the Hindi words in a particular sentence increases(CMI index increases).
- Case 3: Similar trend is visible for the third case as well. The efficiency slightly increases to 78.60% at CMI = 0.25 and then subsides to 64.40% at CMI = 1.00. We notice the same trend in this model as well because of the same reason as mentioned in the previous case i.e. the model has been mainly/predominantly been trained in English so it is harder for it to predict when the Hindi words in a particular sentence increases(CMI index increases). This one uses the XLM-17 model which is trained in 17 languages.

- The common features of all the models are:
  1) Efficiency is highest at CMI = 0.25 or CMI = 0.00 in all cases.
  2) CMI = 1.00 is the point where the accuracy is the lowest.
- Bert is slightly better than mBert. This might be because BERT is trained in Latin Script while mBERT is trained in Devanagari Script and our sentence is wholly in the Latin Script.

References:

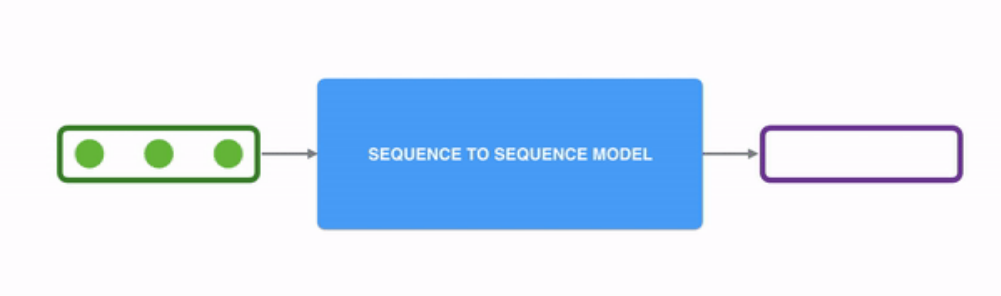
- https://www.kaggle.com/code/jaskaransingh/bert-fine-tuning-with-pytorch
- https://github.com/Kalit31/HASOC-2021/blob/main/English/BERT%20based/finetuned-bert.ipynb
- https://vaclavkosar.com/ml/Tokenization-in-Machine-Learning-Explained#:~:text=Tokenization%20is%20splitting%20the%20input,embedded)%20into%20a%20vector%20space.
- https://towardsdatascience.com/how-to-build-a-wordpiece-tokenizer-for-bert-f505d97dddbb
- https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270

Part 3
Decision Transformer

Explanation of some important concepts:

1) **Transformers:** Transformers were developed to solve the problem of sequence transduction, or neural machine translation. That means any task that transforms an input sequence to an output sequence. This includes speech recognition, text-to-speech transformation, etc.

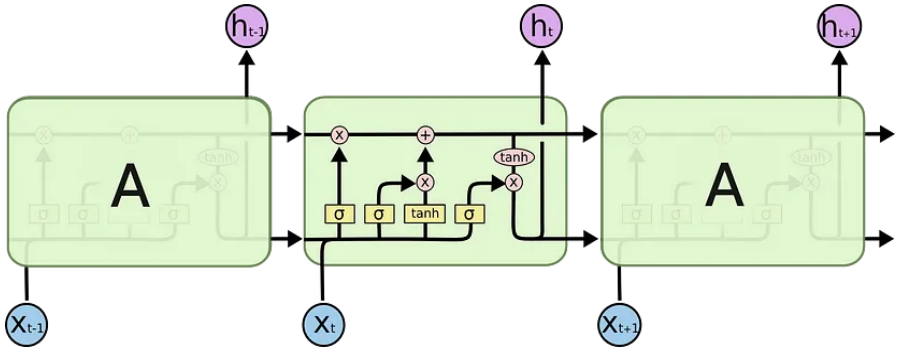
   A transformer model is a neural network that learns context and thus meaning by tracking relationships in sequential data like the words in this sentence.Transformer models apply an evolving set of mathematical techniques, called attention or self-attention, to detect subtle ways even distant data elements in a series influence and depend on each other.



2) **Long-Short Term Memory(LSTM):** LSTMs make small modifications to the information by multiplications and additions. With LSTMs, the information flows through a mechanism known as cell states. In this way, LSTMs can selectively remember or forget things that are important and not so important.

   Each cell takes as inputs x_t (a word in the case of a sentence to sentence translation), the previous cell state and the output of the previous cell. It manipulates these inputs and based on them, it generates a new cell state, and an output.

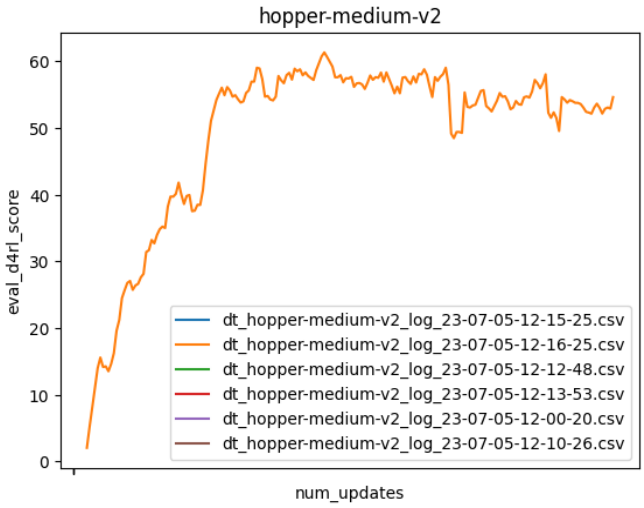
The limitation of LSTM is that : The sequential computation in it inhibits parallelization i.e. we cannot have all outputs together at once because it is sequential. The previous one has to be evaluated before we get the present one.



Results(Using LSTM) :

The RL score is around 59.69 for the LSTM, details are listed below in the table.

{'eval/avg_reward': 1922.4958449093106, 'eval/avg_ep_len': 611.7}
normalized d4rl score: 59.69350095324769
==========
evaluated on env: Hopper-v3
total num of checkpoints evaluated: 1
d4rl score mean: 59.69350
d4rl score std: 0.00000
d4rl score var: 0.00000
==========



In comparison, the transformer from the DT paper has an RL score of around 67.60. Typically because transformers are much stronger at sequence modeling compared to LSTMs, which can suffer from vanishing gradient problems, they have higher RL scores.

Base of code:

- https://github.com/nikhilbarhate99/min-decision-transformer
- https://blogs.nvidia.com/blog/2022/03/25/what-is-a-transformer-model/

References:

- https://github.com/nikhilbarhate99/min-decision-transformer
- https://blogs.nvidia.com/blog/2022/03/25/what-is-a-transformer-model/
- https://jalammar.github.io/illustrated-transformer/
- https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/