Analysis of implementation of different variations of Softmax on CNN for image classification

SAiDL Summer 2023 Induction Assignment 1

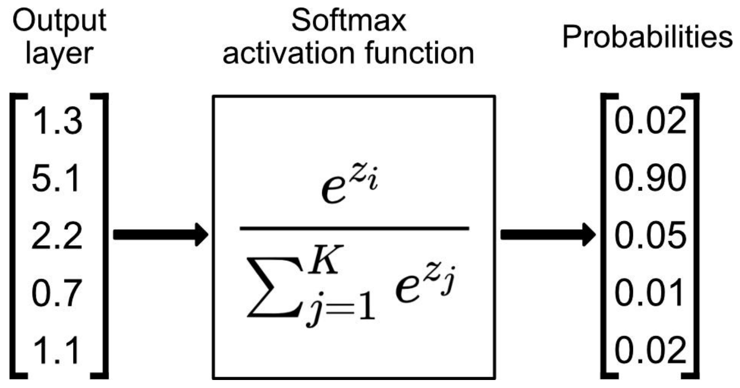
Vimarsh Shah

May 2023

1 Introduction

Neural networks have revolutionized image classification tasks, and the softmax function has become a popular choice for generating probabilistic outputs. However, its time complexity grows linearly with the number of classes, making it computationally expensive for large-scale classification problems.

The need of the softmax function arose when trying to find a close approximation of Arg max (arguments of the maxima) which is differentiable and can be used in back-propagation. Given an unnormalized matrix $[36, 9, 42]$ the argmax function will return 3, as that is the point where the function is maximum at. This function cannot be mathematically found to be differentiable, which is why a very good approximation, softmax is used.


Output layer
1.3
5.1
2.2
0.7
1.1
Softmax activation function
$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
Probabilities
0.02
0.90
0.05
0.01
0.02

Softmax is quite complex to compute and it scales by $O(n)$. To address this issue, alternative softmax implementations have been proposed, such as logsoftmax and Gumbel-Softmax, which aim to reduce the computational complexity while maintaining performance. In this article, we try to implement and understand the performance differences between various implementations.

2 Methodology

In this study, we employ the CIFAR-100 dataset, consisting of 100 classes of images, to compare the performance of different softmax variations in CNN models. First we developed a baseline CNN model with the standard softmax implementation. Then, created additional models using the same architecture but replacing the softmax layer with logsoftmax and Gumbel-Softmax, etc.

2.1 About the dataset - CIFAR-100

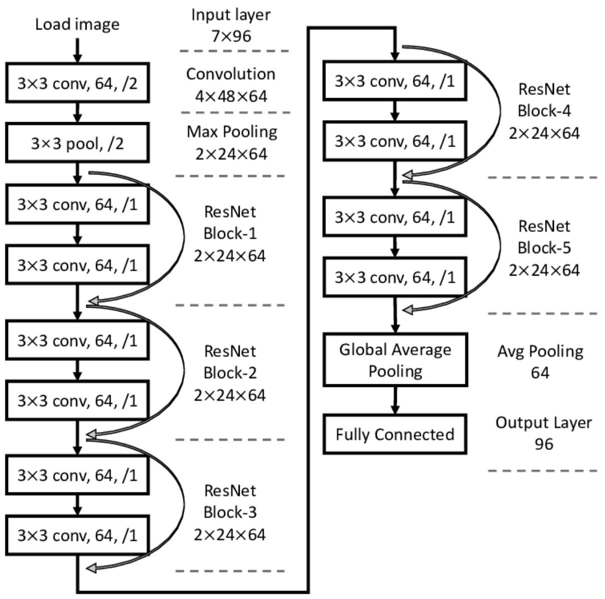
The CIFAR-100 dataset is a popular benchmark dataset for image classification tasks. It is a subset of the larger CIFAR dataset, consisting of 60,000 32x32 color images across 100 different classes. Each class contains 600 images, with 500 images in the training set and 100 images in the test set.

The dataset is organized into a hierarchical structure, with 20 superclasses and 100 fine-grained classes. The superclasses represent broad categories such as aquatic mammals, flowers, or household furniture, while the fine-grained classes provide more specific labels like elephant, sunflower, or table.

To train the model, 50,000 images were used as training dataset with 80:20 train:val split and 10000 images were used for evaluation purpose.

2.2 Architecture of model

The CNN is architected on Resnet12 model. I tried creating my own CNN with several Convolution and activation layers, but accuracy was less compared when compared to already established layer architectures as that of Resnet12.



To this architecture, a last layer was added - an activation layer, being variations of Softmax.

3 Implementations

3.1 Standard Softmax

The standard softmax function is a commonly used activation function in neural networks for multiclass classification problems. Given an input vector, the softmax function transforms it into a vector of probabilities, where each element represents the probability of the input belonging to a particular class. The softmax function is defined as follows:

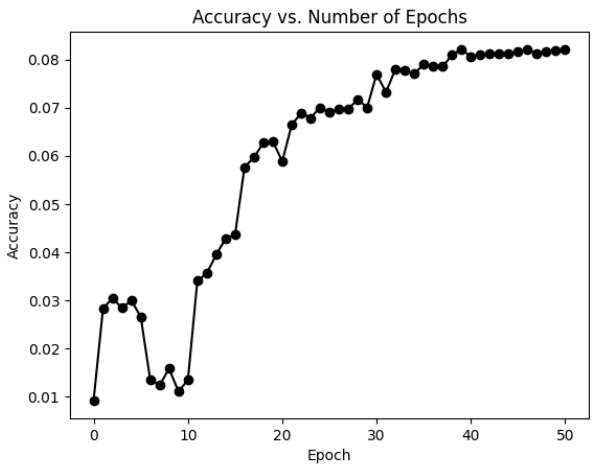
$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$$

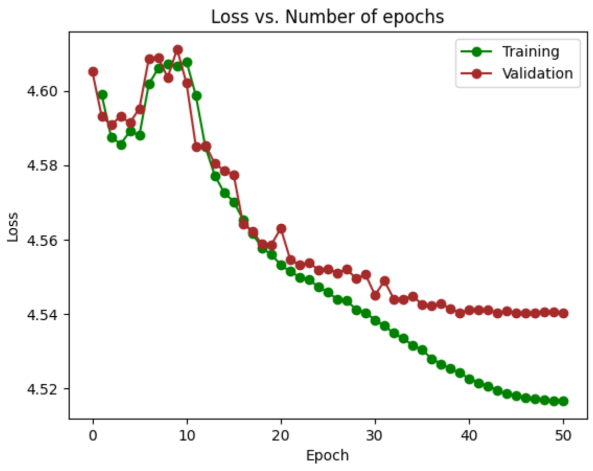
where z_i is the i-th element of the input vector, and N is the total number of classes.

The softmax function computes the exponentiated value of each element in the input vector and normalizes it by the sum of all exponentiated values. This normalization ensures that the output probabilities sum up to 1.

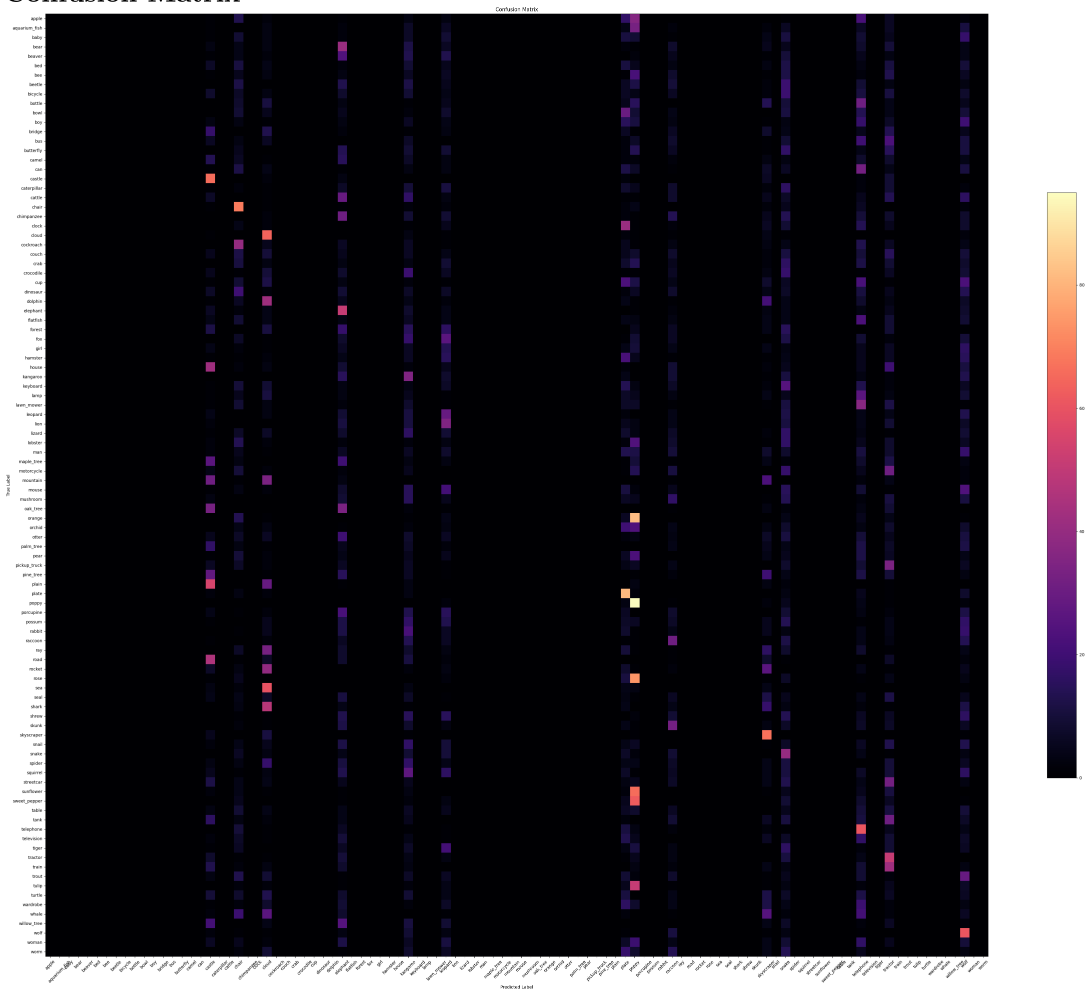
On adding the Softmax layer to the base model, the training results were as follows:

Accuracy	Precision	Recall	F1 Score
8.09%	0.0115	0.0817	0.02





Confusion Matrix



3.2 LogSoftmax

The logsoftmax function is an alternative to the standard softmax function that can provide computational advantages. Instead of directly computing the softmax probabilities, the logsoftmax function calculates the logarithm of the softmax probabilities. The logsoftmax function is defined as follows:

$$\text{LogSoftmax}(z_i) = \log\left(\frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}\right)$$

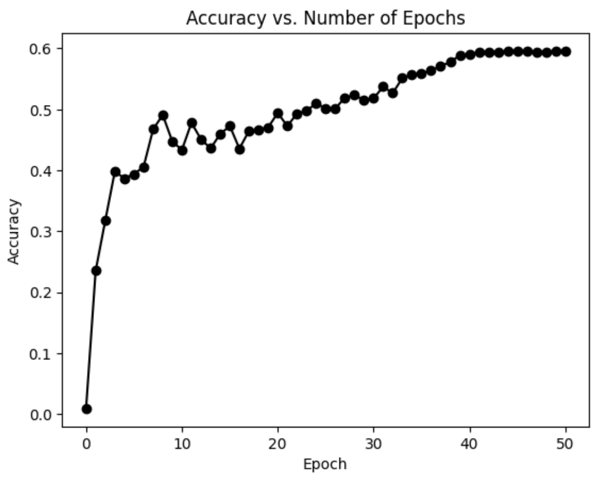
can also be equalized to

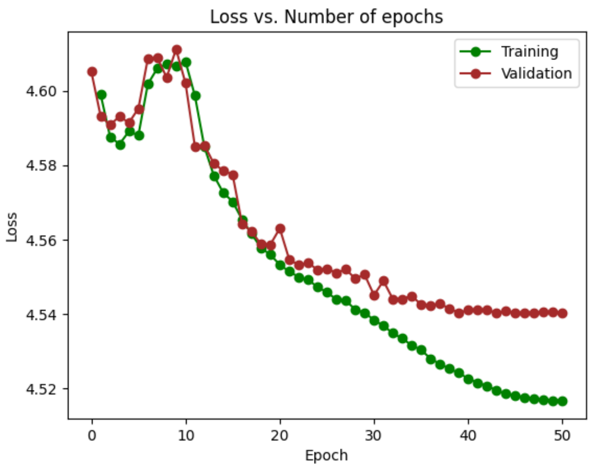
$$\text{logsoftmax}(z_i) = z_i - \log\left(\sum_{j=1}^{N} e^{z_j}\right)$$

The logsoftmax function retains the relative order of the probabilities while reducing the computational complexity. It is often used in combination with the negative log-likelihood loss function, also known as cross-entropy loss, for training neural networks.

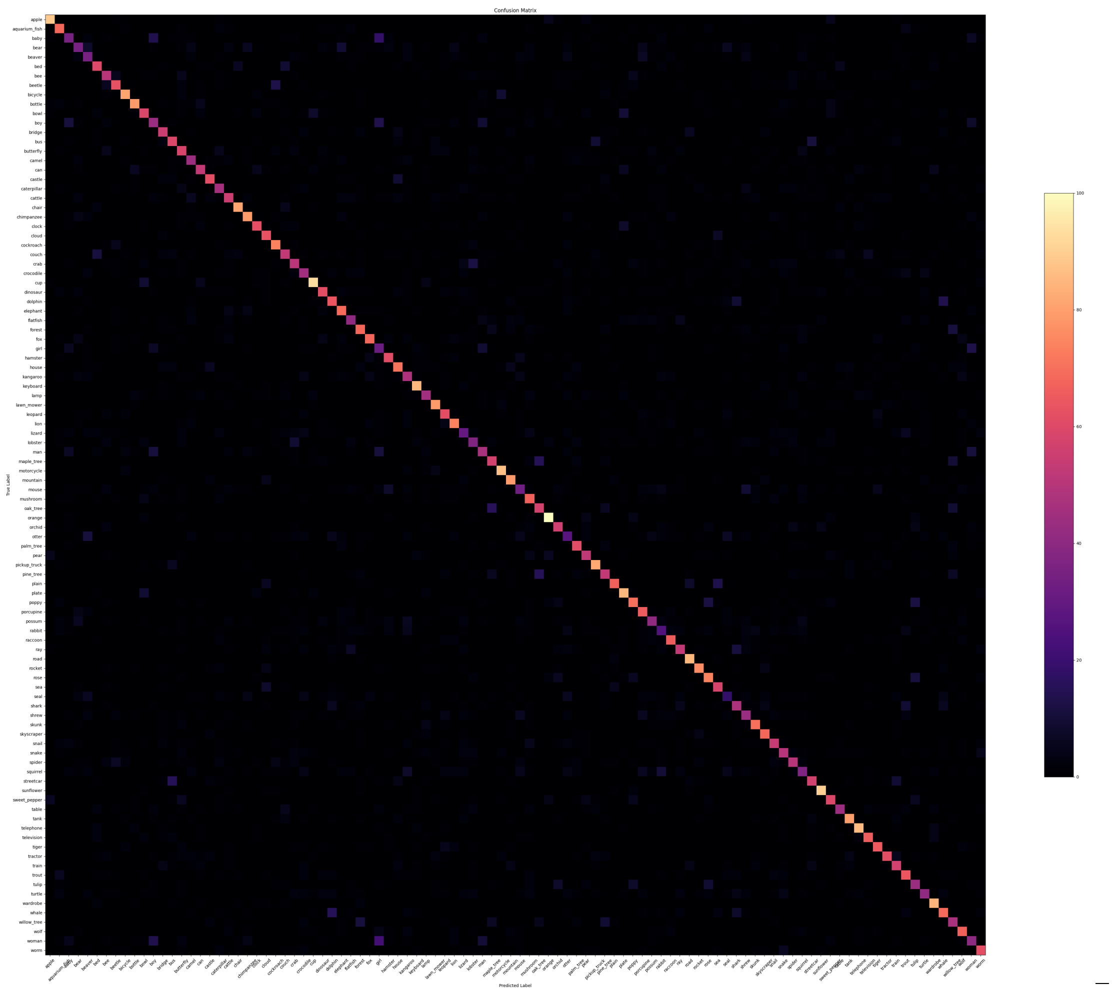
On implementing LogSoftmax instead of standard softmax, the accuracy improved drastically. This seems to be due to the fact that when using normal softmax, the vectors being exponentiated, increase the error by a lot; heavily hindering the performance of the model. But log softmax, simply converts vectors into probability, rather then exponential average. Not only does this reduce complexity but maintains the relative difference, which in this case seems to have helped.

Accuracy	Precision	Recall	F1 Score
60.16%	0.5922	0.5987	0.5931





Confusion Matrix



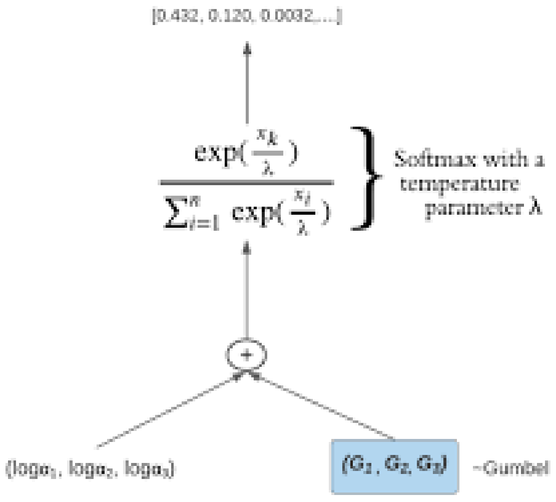
3.3 Gumbel Softmax

Gumbel-Softmax[1] is another alternative to the standard softmax function that introduces a stochastic component. It uses the Gumbel-Max trick to sample from the softmax distribution efficiently. The Gumbel-Softmax function is defined as follows:

$$\text{Gumbel-Softmax}(z_i) = \frac{\exp((\log(z_i) + g_i)/\lambda)}{\sum_{j=1}^{N} \exp((\log(z_j) + g_j)/\lambda)}$$

where z_i is the i-th element of the input vector, g_i is a sample from the Gumbel distribution, and λ is a temperature parameter that controls the smoothness of the distribution. The Gumbel-Softmax function approximates the softmax function while introducing randomness through the Gumbel noise.

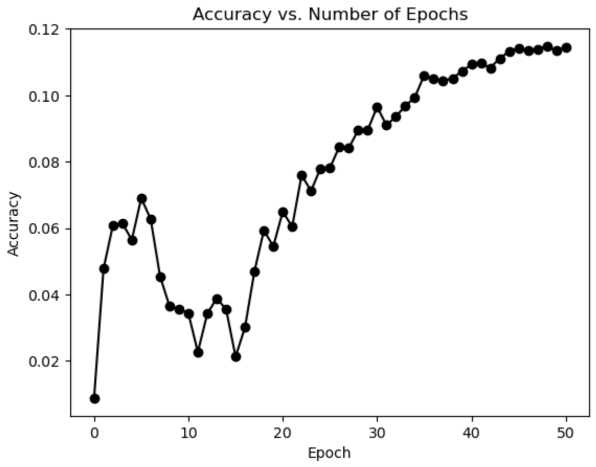
This variation of softmax is just like standard softmax with a temperature term, "reducing" the vectors.

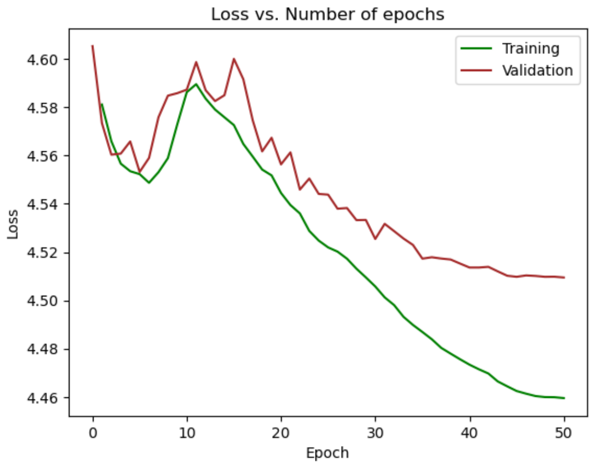

[0.432, 0.120, 0.0032,...]
$\frac{\exp(\frac{x_k}{\lambda})}{\sum_{i=1}^{n}\exp(\frac{x_i}{\lambda})}$
Softmax with a temperature parameter λ
+
$(\log\alpha_1, \log\alpha_2, \log\alpha_3)$
(G_1, G_2, G_3)
~Gumbel

On training the model on this I could see an increase in the accuracy of the model on increasing the value of λ.

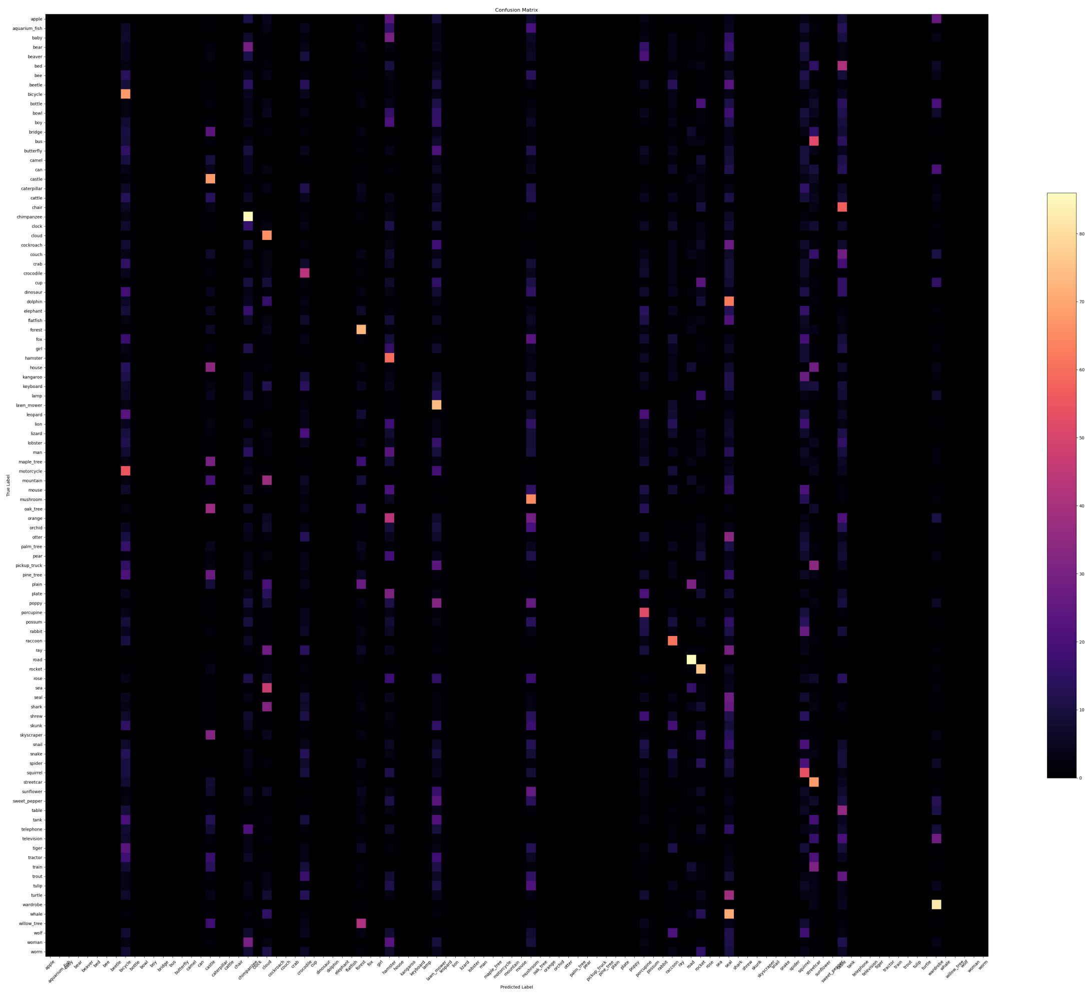
Accuracy	Precision	Recall	F1 Score
11.83%	0.0244	0.1168	0.396

The above accuracy was with the value of λ= 3, on having value ¡=1 the accuracy didn't even reach 8%. Overall the learning rate wrt the number of epochs didn't change much.





Confusion Matrix



– For λ = -1.5:

Accuracy	Precision	Recall	F1 Score
8.81%	0.0155	0.0888	0.0258

Gumbel like softmax gives poor performance. The scaling factor of lambda, does help as it reduces the exponents (explained later), but this approach does not scale well with classes.

3.4 Log Gumbel Softmax

The Log Gumbel-Softmax function is derived from the Gumbel-Softmax function by taking the logarithm of the probabilities. The Gumbel-Softmax function introduces stochasticity through Gumbel noise and approximates the softmax function. The Log Gumbel-Softmax function extends this by applying the logarithm.

Let's denote the input vector as z and the temperature parameter as λ. The Log Gumbel-Softmax function is defined as follows:

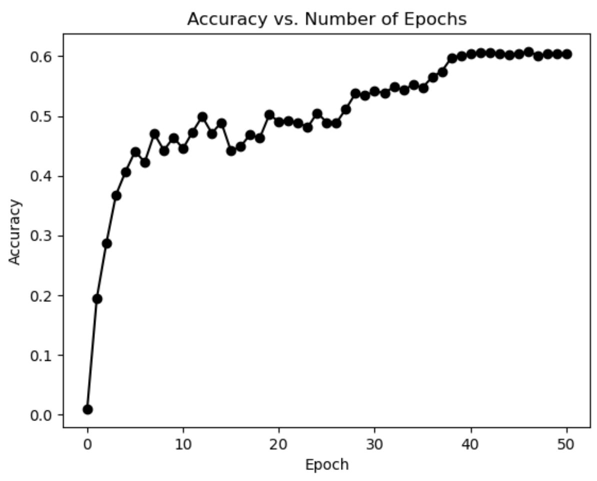
$$\text{LogGumbelSoftmax}(z_i) = \log\left(\frac{\exp((\log(z_i) + g_i)/\lambda)}{\sum_{j=1}^{N} \exp((\log(z_j) + g_j)/\lambda)}\right)$$

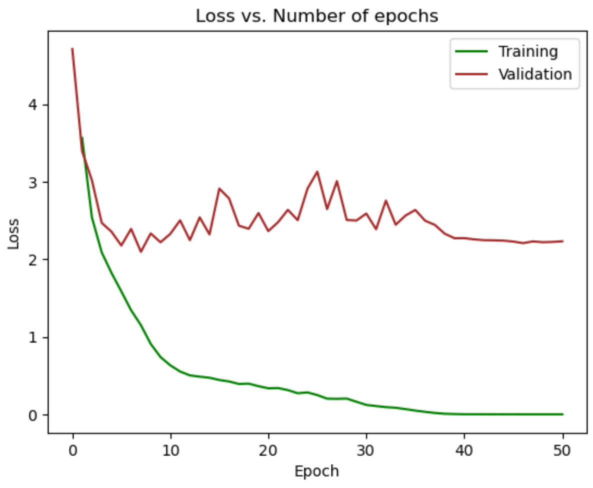
where z_i is the i-th element of the input vector, g_i is a sample from the Gumbel distribution, and N is the total number of classes.

The Gumbel noise introduces randomness and allows for the relaxation of discrete distributions into continuous relaxations. The temperature parameter λ controls the smoothness of the distribution and the trade-off between exploration and exploitation.

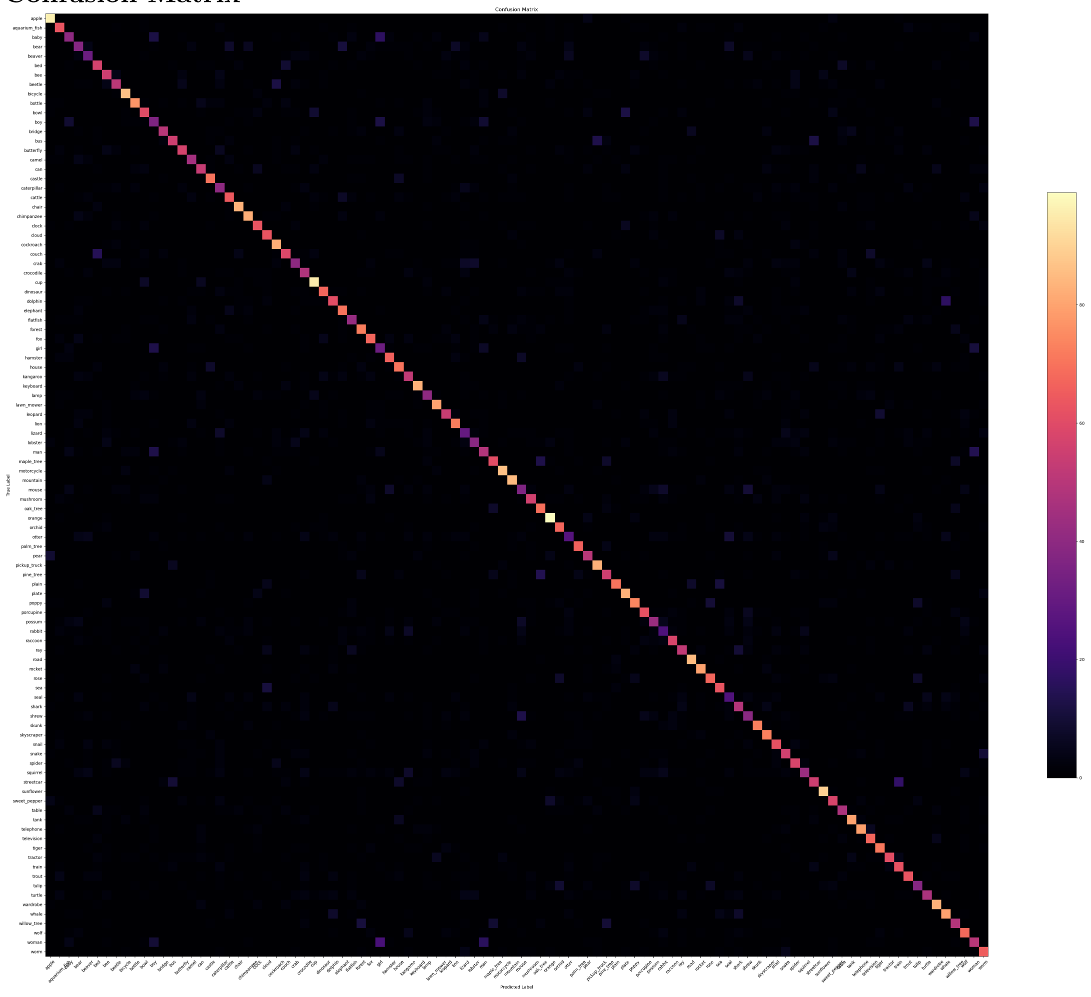
For λ = 3:

Accuracy	Precision	Recall	F1 Score
61.04%	0.6033	0.6091	0.6035





Confusion Matrix



–

For λ = -1.5:

Accuracy	Precision	Recall	F1 Score
59.44%	0.5846	0.5899	0.5850

–

3.5 Other attempts

What if instead of have e^z we have 2^z in the softmax function. This does not change a lot in terms of mathematical operations, but does help in computational scaling. As by the use of bitwise operator computers can quickly calculate 2^z. This does not seem to do anything, as it is the same computation. Although on using low level operations, by using bitwise operations, such 2_i^z operations can be optimized.

I tried various other variations of softmax-like operations:
- $exp(P[Z < z_i])$
- $exp(sin(z_i))$
- Sparsemax[2]
- Taylor approximation of e^x for k-terms

None of these gave decent accuracies. All were around that of Softmax, with similar functioning.

The only variation that gave decent predictions was that log of taylor softmax. This would evaluate to almost the same perfomance as that of LogSoftmax, as essentially performs the same operation. Another novel method was also tried, which dropped the lowest k (= 10%) of the terms in the output matrix, followed by the activation layer. This resulted in not much but 2-3% improvement in model performance on benchmarks.

4 Evaulation

4.1 Reason for low performance of Softmax, when compared to others

To compare the accuracy of softmax and log softmax, let's consider a 1D array as an example: [13, 8, 2]. We will calculate the probabilities using both softmax and log softmax and analyze the differences.

Softmax

The softmax function transforms the input vector into a probability distribution. Let's calculate the softmax probabilities for the given array.

$$\text{Softmax}([13,8,2]) = \left[\frac{e^{13}}{e^{13}+e^{8}+e^{2}}, \frac{e^{8}}{e^{13}+e^{8}+e^{2}}, \frac{e^{2}}{e^{13}+e^{8}+e^{2}}\right]$$

Calculating the exponential values:

$$\text{Softmax}([13,8,2]) = \left[\frac{442413.39}{442413.39+2980.96+7.39}, \frac{2980.96}{442413.39+2980.96+7.39}, \frac{7.39}{442413.39+2980.96+7.39}\right]$$

Normalizing the values:

$$\text{Softmax}([13,8,2]) \approx [0.999, 0.001, 0]$$

The softmax probabilities indicate that the first element has a high probability (approximately 99.9%) of belonging to the class, while the second element has a very low probability (approximately 0.1%), and the third element has an almost negligible probability.

LogSoftmax

The log softmax function calculates the logarithm of the softmax probabilities. Let's calculate the log softmax probabilities for the same array.

$$\text{LogSoftmax}([13,8,2]) = \left[13 - \log(e^{13}+e^{8}+e^{2}), 8 - \log(e^{13}+e^{8}+e^{2}), 2 - \log(e^{13}+e^{8}+e^{2})\right]$$

Calculating the exponential values:

$$\text{LogSoftmax}([13,8,2]) = [13 - \log(442413.39+2980.96+7.39), 8 - \log(442413.39+2980.96+7.39), 2 - \log(442413.39 +$$

Normalizing the values:

$$\text{LogSoftmax}([13,8,2]) \approx [13-13.1, 8-13.1, 2-13.1] \approx [-0.1, -5.1, -11.1]$$

The log softmax probabilities indicate that the first element has a relatively high value (-0.1), the second element has a lower value (-5.1), and the third element has the lowest value (-11.1).

Pytorch while using logsoftmax, takes the (-) negative logarithm, making all values (+) positive. The reason is that softmax returns values between (0,1). On taking log on this output, the outputs are between (-∞, 0), which is why it negates the output.

Reasoning

The difference in accuracy between softmax and log softmax arises from the nature of the functions and the properties they preserve.

Softmax calculates the exponential values of the input vector, which can lead to significant differences in magnitudes. In our example, the element with the highest value (13) has a much larger exponential value compared to the other elements. This exponential amplification causes the softmax probabilities to be heavily skewed towards the highest value, resulting in a very high probability for the first element and extremely low probabilities for the other elements.

On the other hand, log softmax calculates the logarithm of the softmax probabilities. Taking the logarithm dampens the differences in magnitudes and compresses the range of values. In our example, the log softmax probabilities are relatively close to each other (-0.1, -5.1, -11.1), indicating a more balanced distribution of probabilities among the elements.

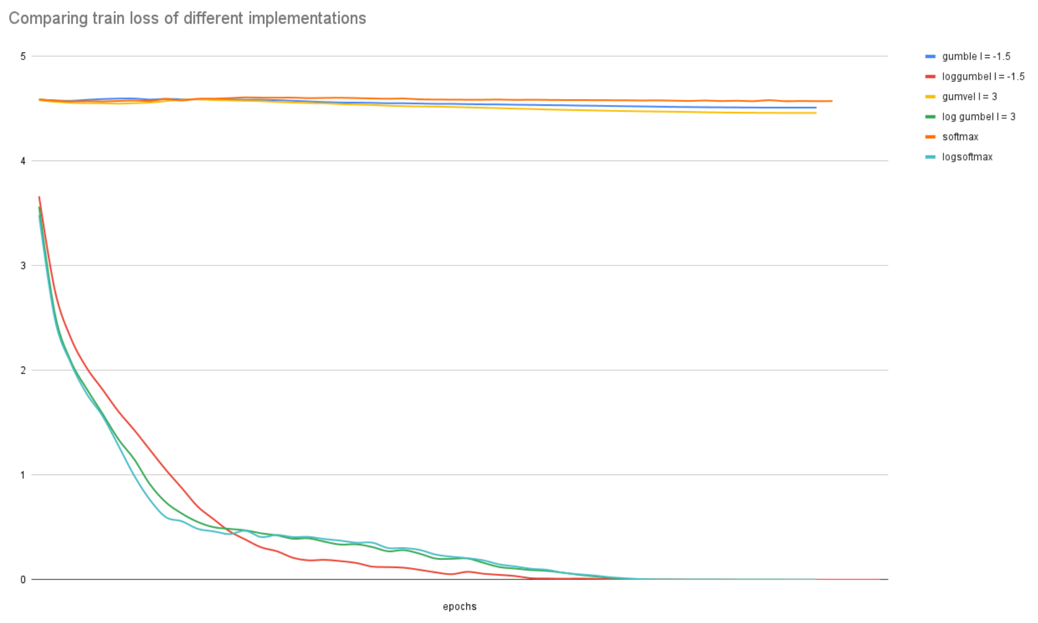
The log softmax function is therefore preffered over softmax when dealing with models that have a large output space like image classification of larger datasets. By taking the logarithm, log softmax reduces the chances of numerical overflow or underflow that can occur with softmax, especially when dealing with large or small exponentiated values.

In terms of accuracy, log softmax is much more balanced and probabilities are more useful when compared to softmax, especially in scenarios where there are significant differences in the magnitudes of the input vector elements.

4.2 Using 20 Super Classes

When I trained the model on just the 20 super classes, the accuracy with activation layer being Softmax for 50 epochs was 45% , which decreases drastically on increasing it to 100 classes. Further showing that the complexity increases drastically with increasing the number of classes.

4.3 Comparing different variations



Based on different schemes of evaluation, it is evident that the performance of Softmax and its variations are much worse than its log implementations.

It becomes evident that the log implementations generally outperform their normal counterparts in certain scenarios. Here are some reasons why log variations are often considered better:

1. Numerical Stability: LogSoftmax addresses the issue of numerical stability that can arise with regular softmax when dealing with large or small exponential values. By taking the logarithm, LogSoftmax reduces the chances of numerical overflow or underflow. This is particularly beneficial when working with models that have a large output space or when dealing with data that has significant variations in magnitudes. In our case CIFAR 100 had a relatively large number of classes (100) resulting in poor performance.

2. Balanced Probabilities: LogSoftmax produces a more balanced distribution of probabilities compared to softmax. In softmax, exponential amplification of values can lead to highly skewed probabilities, where one element dominates while others are assigned extremely low probabilities. LogSoftmax, on the other hand, compresses the range of values, resulting in more evenly distributed probabilities.

As seen in 4.1 usage of softmax skews the values too much, making it difficult to adjust the model by very small percentages.

3. The Dataset: CIFAR 100 is a low resolution dataset (32x32) with improper lighting and not proper filtering. It also has a high number of output classes. When the model has less data to work with (just 32x32 pixels) the features it extracts are not that different from each other. When we use softmax as activation layer, due to there not being a substantiated difference between output vectors, it does not perform well.

5 Conclusion

In this report, we explored different variations of softmax and how they impact image classification task. The analysis revealed the massively better performance of log variations of softmax functions, and we took a look on the reasoning behind on why it performed that way. In summary, log transformations provide a more measurable and meaningful probability values, facilitating better model performance.

Overall, the findings highlight the differences in activation layers and their effects on image classification task.

Further research could focus on verifying the hypotheses of number of classes by measuring the impact on different datasets, corresponding to different domains. By exploring the applicability of this function, novel variations could be developed that solve challenges pertaining to that domain.

Other references[3][4][5][6]

References

[1] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. 2017.

[2] André F. T. Martins and Ramón Fernandez Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. *CoRR*, abs/1602.02068, 2016.

[3] Vinod Nair Alex Krizhevsky and Geoffrey Hinton. Cifar-100 dataset, 2009.

[4] Ifueko Igbinedion, Derek Phillips, and Daniel Lévy. Fast softmax sampling for deep neural networks. *CS231n*, 2017.

[5] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables, 2017.

[6] Ali Tabak. 5 modelling approaches for image classification. 2020.