SAiDL Summer Induction Assignment

Harshvardhan Mestha

28th May, 2023

Contents

Impact of the Softmax activation function and its variants for image classification on the CIFAR-100 dataset

Harshvardhan Mestha

May 28, 2023

1 Introduction

The Softmax function is commonly used as an activation function for multiclass classification. However the function is very costly to compute as the number of classes increase. In addition to this, the function is most frequently added in the final layer of the model, which can have drastic effects on the accuracy and other metrics of the model due to the very nature of the Softmax function, which we shall see later. The above reasons warrant the exploration of other alternatives and variants of the Softmax. I have analysed 4 variants namely - log-softmax, log-Taylor softmax log-Gumbel softmax, and Gumbel softmax by conducting image classification on the CIFAR-100 dataset.

2 The Softmax Function and its variants

This section has an overview of the activation functions being compared.

2.1 Softmax

The softmax function is computed as follows:

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

where $\mathbf{z} = (z_1, z_2, \cdots, z_N) \in R^N$ and $i = 1, 2, \cdots, N$ and N is the number of classes.

This function has a time complexity of O(N) meaning it takes longer to compute as the number of classes increase. This is due to the long time taken for the denominator to compute.
Let us take an elementary example to understand how the function works: If we take an input tensor of $\mathbf{z}$ = [8,7,1] the output tensor is softmax($\mathbf{z}$) = [0.731,0.268,0.001]. The function gives a very high weight to the largest value and gives extremely small weights to the non-largest values, even though the difference between the largest and the non-largest value is not very high.

2.2 Log Softmax

The log softmax function is computed as follows:

$$log\_softmax(\mathbf{z}) = \log \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_i}}$$

where $\mathbf{z} = (z_1, z_2, \cdots, z_N) \in R^N$ and $i = 1, 2, \cdots, N$ and N is the number of classes.

This function has a time complexity of O(N) making it faster than the standard softmax
Let us take an elementary example to understand how the function works: If we take an input tensor of $\mathbf{z}$ = [8,7,1] the output tensor is log_softmax($\mathbf{z}$) = [-0.313,-1.317,-6.908]. The log softmax is more proportionate when assigning weights as compared to the standard softmax. This is due to the numerators becoming z_i as $\log e^{z_i} = z_i$ and the denominator is constant, so the differences in the values of the tensor are not amplified exponentially.

2.3 Gumbel Softmax

The Gumbel softmax function is computed as follows:

$$gumbel\_softmax(\mathbf{z}) = \frac{e^{z_i/\lambda}}{\sum_{j=1}^{N} e^{z_i/\lambda}} + gumbel\_noise$$

where $\mathbf{z} = (z_1, z_2, \cdots, z_N) \in R^N$ and $i = 1, 2, \cdots, N$ and N is the number of classes. *gumbel_noise* is random noise sampled from the Gumbel distribution. λ is the temperature parameter, can also be denoted by tau (τ)

This variant allows to scale the input tensor with the temperature parameter making the gumbel softmax attribute weights better than standard softmax.

2.4 Log Gumbel Softmax

The Log Gumbel softmax function is computed as follows:

$$log\_gumbel\_softmax(\mathbf{z}) = \log(\frac{e^{z_i/\lambda}}{\sum_{j=1}^{N} e^{z_i/\lambda}} + gumbel\_noise)$$

where $\mathbf{z} = (z_1, z_2, \cdots, z_N) \in R^N$ and $i = 1, 2, \cdots, N$ and N is the number of classes. *gumbel_noise* is random noise sampled from the Gumbel distribution. λ is the temperature parameter, can also be denoted by tau (τ)

This function is simply the logarithm of the standard gumbel_softmax allowing us to scale tensors and have a proportional weight distribution.

2.5 Log Taylor Softmax

The Log Taylor softmax function is computed as follows:

$$log\_taylor\_softmax(\mathbf{z}) = \log(\frac{1 + z_i + 0.5{z_i}^2}{\sum_{j=1}^{N} 1 + z_i + 0.5{z_i}^2})$$

where $\mathbf{z} = (z_1, z_2, \cdots, z_N) \in R^N$ and $i = 1, 2, \cdots, N$ and N is the number of classes.

This variant of the softmax uses the second order Taylor approximation of the exponential, but takes the natural logarithm of the output. This results in an increase in the speed of computing the denominator, due to the Taylor approximation, and also proportionally assigns the weights similar to Log Softmax

3 Evaluating the various models

This section will cover the various evaluation metrics for the different activations.

All the models use the Resnet9 Convolutional Neural Network architecture.Note that the models use MaxPool2d instead of AvgPool.

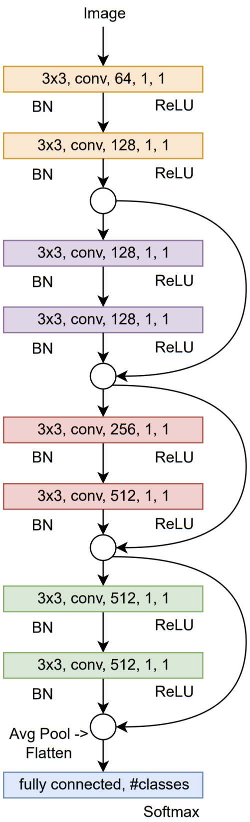


Figure 1: The Resnet9 CNN architecture.

Common parameters between the models:

- Number of epochs = 100
- Maximum Learning Rate = 0.01
- Gradient Clipping = 0.1
- Weight Decay = 0.0001
- Manual random seed (for torch.manual_seed) = 43
- Dropout(0.25) for classification layer
- All the models were run using the T4 GPU runtime on Google colab.

The CIFAR-100 dataset: The dataset has not been modified - train-test split is 50000 images for training and 10000 test images. The classification has been conducted on the 100 subclasses. The dataset has 60000 images (32*32 pixels, 3 channel colour).
Note: The temperature for gumbel_softmax and log_gumbel_softmax was $\lambda = 5$, and better metrics can be obtained testing other values of tau. This report does not analyse the nature of the gumbel_softmax and therefore has only 1 particular example.

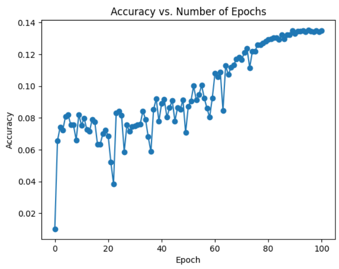
3.1 Accuracy

Activation function	Accuracy (%)
softmax	13.37
log_softmax	59.16
gumbel_softmax	14.84
log_gumbel_softmax	55.87
log_taylor_softmax	49.19

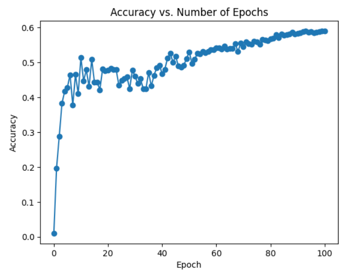
Table 1: Accuracies for the various activation functions.

There is a clear increase in accuracy when we apply the logarithm to the respective standard function.

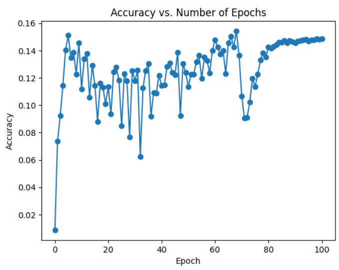
The standard softmaxes can be useful when we want to amplify one class disproportionately over the others, when an ambiguity arises between one or more classes. However, in a classification task with a large number of classes such as the CIFAR-100, the standard softmaxes lose accuracy which is attributed to the fact that the model is unable to distinguish between classes that are similar in appearance but have different labels, due to the exponential amplification of small differences.



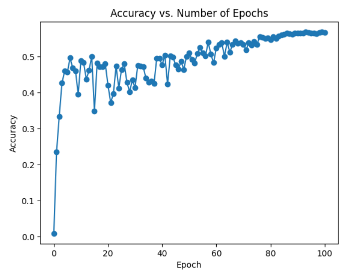
softmax



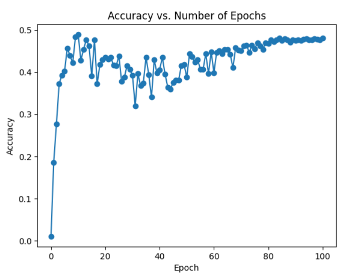
log_softmax



gumbel_softmax



log_gumbel_softmax



log_taylor_softmax

We can observe that non-logarithmic have a very unstable gradient descent pathway as seen by the large fluctuations in accuracy as the epoch number rises. But all models eventually converge by the time the last epoch is reached.

3.2 Precision

Activation function	Precision
softmax	0.0280
log_softmax	0.5891
gumbel_softmax	0.0330
log_gumbel_softmax	0.5646
log_taylor_softmax	0.5107

Table 2: Precision for the various activation functions.

Here the trend is similar to accuracy where the logarithmic activation outperform their non logarithmic counterparts.

3.3 Recall

Activation function	Recall
softmax	0.1343
log_softmax	0.5885
gumbel_softmax	0.1496
log_gumbel_softmax	0.5632
log_taylor_softmax	0.4782

Table 3: Recall for the various activation functions.

Here the trend is similar to accuracy where the logarithmic activation outperform their non logarithmic counterparts.

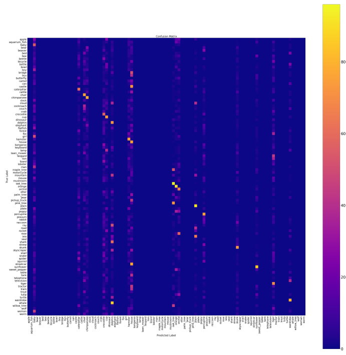
3.4 F1 Score

Activation function	F1 score
softmax	0.0457
log_softmax	0.5867
gumbel_softmax	0.0533
log_gumbel_softmax	0.5617
log_taylor_softmax	0.4760

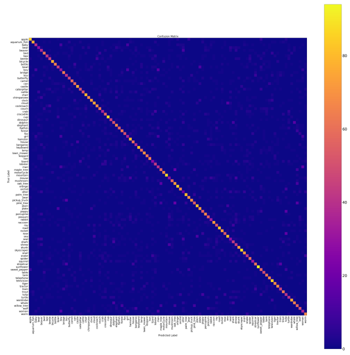
Table 4: F1 scores for the various activation functions.

Here the trend is similar to accuracy where the logarithmic activation outperform their non logarithmic counterparts.

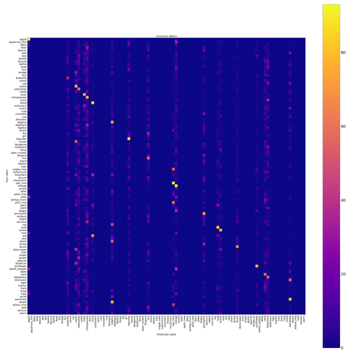
3.5 Confusion Matrices



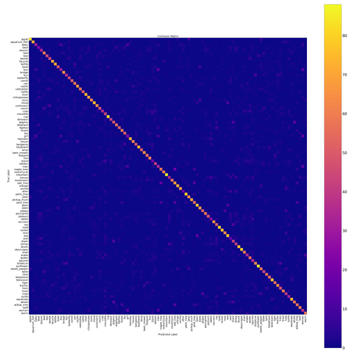
softmax



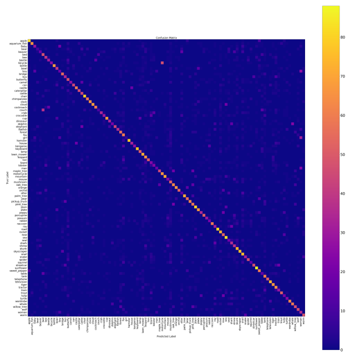
log_softmax



gumbel_softmax



log_gumbel_softmax



log_taylor_softmax

We observe that the logarithmic activations produce classifiers with much more well defined confusion matrices.

3.6 Epoch time

Epoch time has been calculated as:

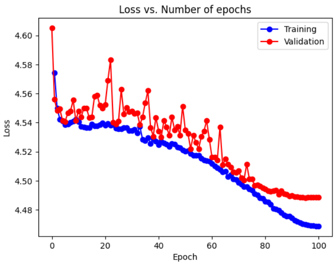
$$\text{epoch time} = \frac{\text{wall time after 100 epochs}}{100}$$

Activation function	Epoch time (seconds)
softmax	19.95
log_softmax	20.33
gumbel_softmax	19.64
log_gumbel_softmax	21.07
log_taylor_softmax	20.59

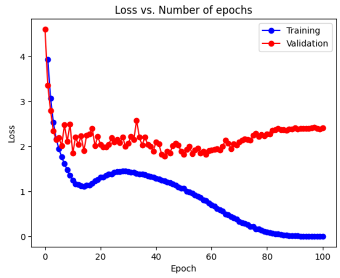
Table 5: Epoch time for the various activation functions.

We observe that there is a minor time penalty associated with taking the logarithm, however we see log_softmax is the fastest logarithmic softmax, and gumbel_softmax is faster than the standard softmax.
However the models having a logarithmic softmax have finished training around 25 epochs earlier than the non-logarithmic softmaxes, so they train around 20% faster since they need lesser epochs to reach their minimal loss.

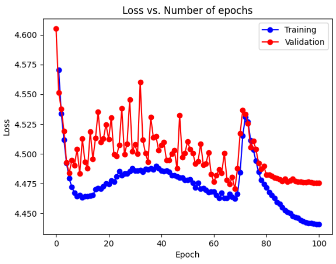
3.7 Loss v/s epochs



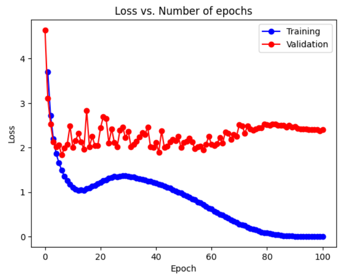
softmax



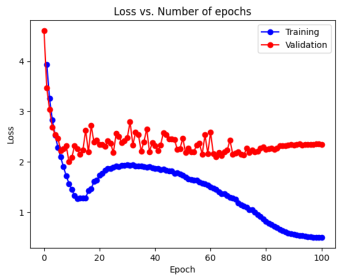
log_softmax



gumbel_softmax



log_gumbel_softmax



log_taylor_softmax

We can see that the training and validation losses are more closely associated with each other and diverge much slowly in the non-logarithmic softmaxes. The logarithmic softmaxes reach their divergence point quicker and the loss stabilizes around the 75^{th} epoch, meaning the models have finished training earlier.

In addition to finishing training in lesser epochs, the logarithmic softmaxes also minimize the loss more than their non-logarithmic counterparts.

4 Conclusions

To conclude, I have found that taking the logarithm of the standard softmax variations outperform the non-logarithmic versions in every metric analysed here. However these results may not hold true for datasets with lower number of classes such as the MINST dataset where the softmax function benefits the model by reducing uncertainty. gumbel_softmax is more performant than the standard softmax, and its efficacy can be improved by optimising the λ or temperature parameter. The log-Softmax is the most performant of the logarithmic softmaxes, however the performance of the log_taylor_softmax can be adjusted by adjusting the order of the Taylor polynomial to which we calculate, which requires further examination.

5 Sources/References

Kunal Banerjee et al, "Exploring Alternatives to Softmax Function", 2020, https://arxiv.org/pdf/2011.11538.pdf

Figure 1 from: Chandramouli Rajagopalan et al, "Deep learning in a bilateral brain with hemispheric specialization", 2023, https://arxiv.org/pdf/2209.06862.pdf

Youngkyu Hong et al, Disentangling Label Distribution for Long-tailed Visual Recognition, 2021, https://arxiv.org/pdf/2012.00321v2.pdf

Introduction to Softmax for Neural Network, https://www.analyticsvidhya.com/blog/2021/04/introduction-to-softmax-for-neural-network/

Softmax: Multiclass Neural Networks, https://www.turing.com/kb/softmax-multiclass-neural-networks

Train your image classifier model with PyTorch, https://learn.microsoft.com/en-us/windows/ai/windows-ml/tutorials/pytorch-train-model

Image Classification in a Nutshell: 5 Different Modelling Approaches in PyTorch with CIFAR100, https://medium.com/@alitbk/image-classification-in-a-nutshell-5-different-modelling-approaches-in-pytorch-with-cifar100-8f690866b373

Softmax and Uncertainty, https://towardsdatascience.com/softmax-and-uncertainty-c8450ea7e064

https://github.com/Harshvardhan-Mestha/SAiDL_Assignment