COL870: Assignment 1 | Report

Ankit Garg 2017EE10439
Devang Mahesh 2017EE10093

Link to the Model Weight Folder:
https://drive.google.com/drive/folders/1ohoIx0j60WGXgmx-W5f3-ZEgLr9N_uf_?usp=sharing

Part 1

ResNet over Convolutional Networks and different Normalization schemes

We built a generalized ResNet architecture of depth 6*n + 2 , and r outputs. We trained this model on the CIFAR-10 dataset, with n = 2 and r = 10, and reported the results. We experimented with various normalization techniques.

The train / validation / test split was 40k / 10k / 10k images. We used SGD optimizer, in all the experiments, with weight decay of 0.0001.

Part 1.1
Image Classification using Residual Network

Normalization

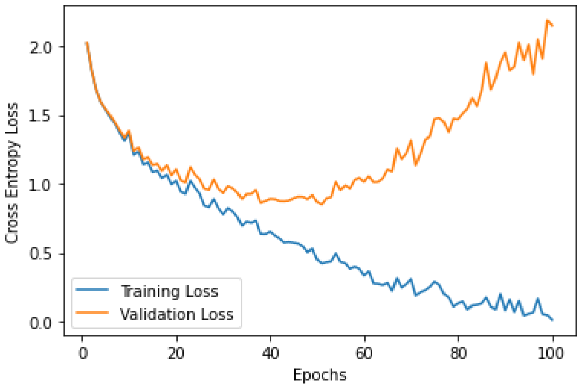
Without any normalization

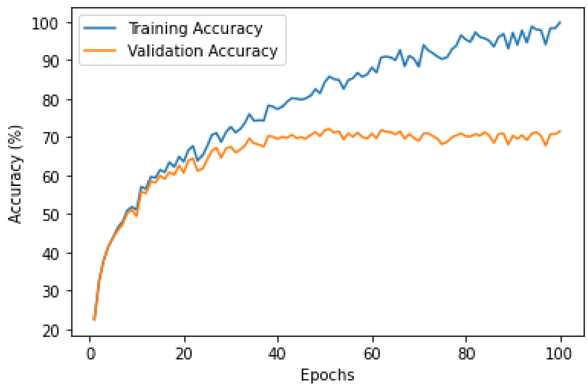
Hyperparameters

Optimizer	SGD
Learning Rate	0.001
Momentum	0.9
Weight Decay	0.0001
Batch Size	128

Plots

The error and accuracy plots are obtained as shown:-





Regularization

- **Early stopping**
  Early stopping at epoch = 49

- **Weight Decay**
  We use weight decay = 0.0001

Results

Data	Accuracy	Macro F1	Micro F1
Training	85.78%	85.80%	93.52%
Validation	72.09%	72.04%	82.61%
Test	71.74%	71.73%	82.26%

Part 1.2 : Impact of Normalization

We implemented various normalizations, from scratch in pytorch :-

(a) Batch Normalization
(b) Instance Normalization
(c) Batch-Instance Normalization
(d) Layer Normalization
(e) Group Normalization

The statistics of No Normalization variant has been mentioned in the previous part. We report variant wise statistics, and then we perform the comparative study. The normalization layers are applied just before the activation in all these cases.

1.2.1 - 1.2.3

- ## BN Variant

Normalization

This uses batch normalization

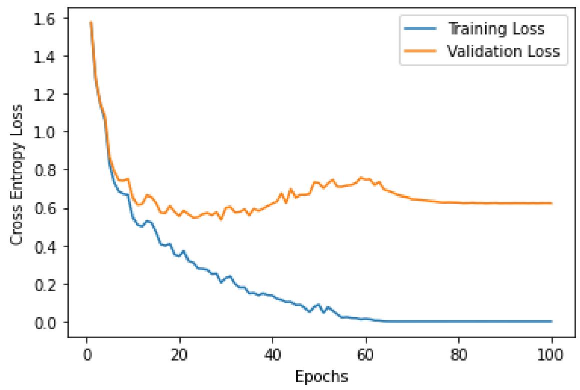
Hyperparameters

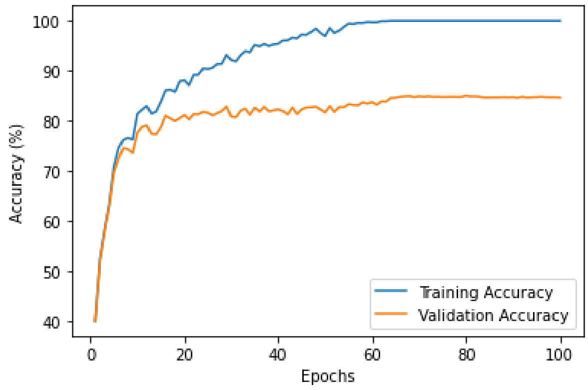
Optimizer	SGD
Learning Rate(initial)	1
Momentum	0.9
Weight Decay	0.0001
Batch Size	128

It starts with a high learning rate of 1. After every epoch, the learning rate becomes 0.96 times the previous learning rate.

Plots

The error and accuracy plots are obtained as shown:-





Regularization

- **Early stopping**
  Early stopping at epoch = 40

- **Weight Decay**
  We use weight decay = 0.0001

Results

Data	Accuracy	Macro F1	Micro F1
Training	93.19%	93.17%	96.48%
Validation	82.87%	82.78%	89.44%
Test	81.39%	81.28%	89.48%

- ## IN Variant

Normalization

This uses instance normalization

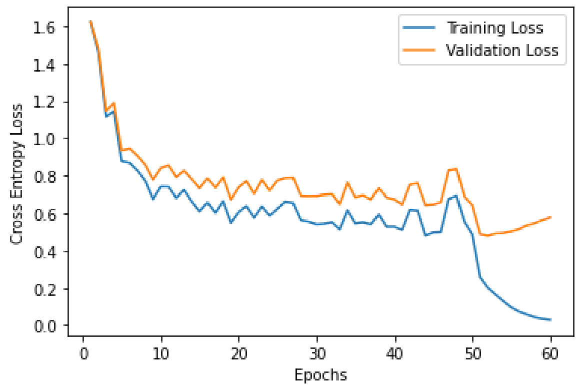
Hyperparameters

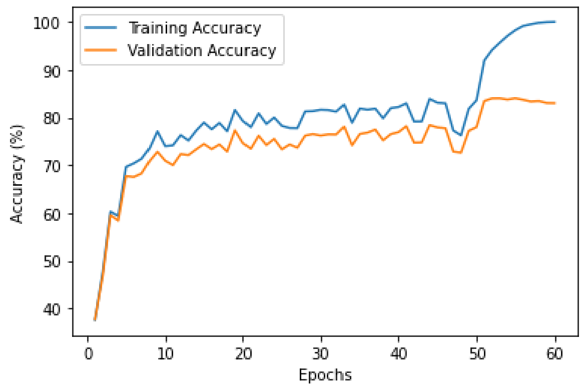
Optimizer	SGD
Learning Rate (initial)	1
Momentum	0.9
Weight Decay	0.0001
Batch Size	128

It starts with a high learning rate of 1. After 50 Epochs, we change it to 0.1

Plots

The error and accuracy plots are obtained as shown:-





Regularization

- **Early stopping**
  Early stopping at epoch = 52

- **Weight Decay**
  We use weight decay = 0.0001

Results

Data	Accuracy	Macro F1	Micro F1
Training	94.05%	94.08%	97.23%
Validation	83.99%	84.06%	90.19%
Test	83.35%	83.43%	89.79%

- ## BIN Variant

Normalization

This uses batch - instance normalization

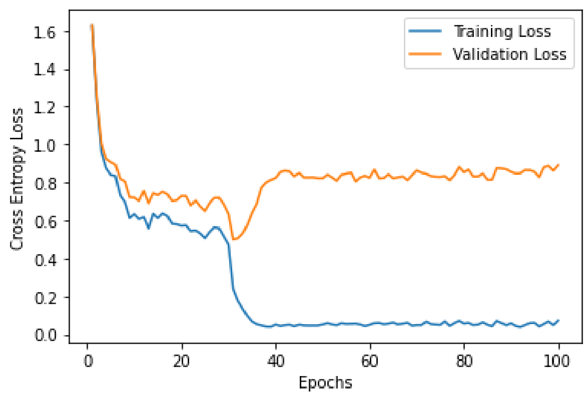
Hyperparameters

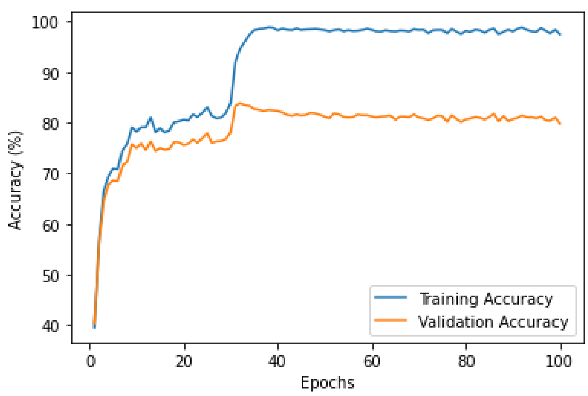
Optimizer	SGD
Learning Rate (initial)	1
Momentum	0.9
Weight Decay	0.0001
Batch Size	128

It starts with a high learning rate of 1. After 30 Epochs, we change it to 0.1

Plots

The error and accuracy plots are obtained as shown:-





Regularization

- **Early stopping**
  Early stopping at epoch = 35

- **Weight Decay**
  We use weight decay = 0.0001

Results

Data	Accuracy	Macro F1	Micro F1
Training	92.12%	92.14%	96.31%
Validation	83.31%	83.38%	90.32%
Test	82.98%	83.00%	89.78%

- ## LN Variant

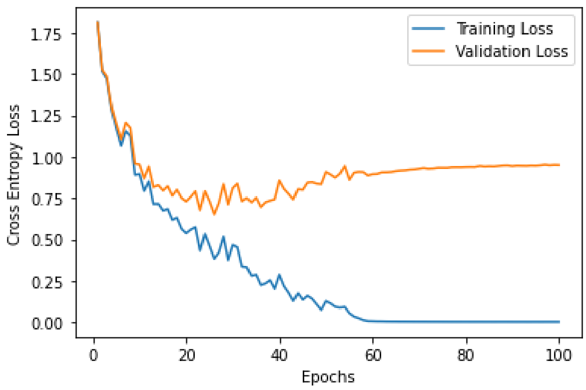
Normalization
This uses layer normalization

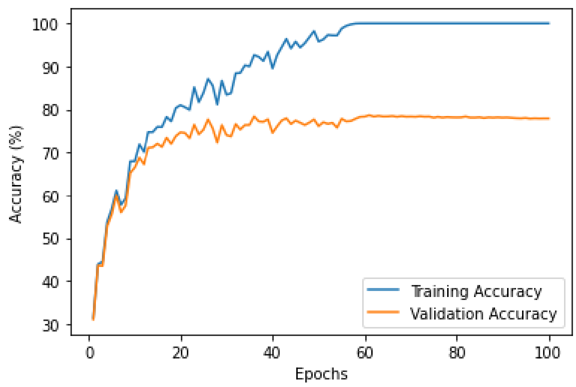
Hyperparameters

Optimizer	SGD
Learning Rate	0.01
Momentum	0.9
Weight Decay	0.0001
Batch Size	128

Plots

The error and accuracy plots are obtained as shown:-





Regularization

- **Early stopping**
  Early stopping at epoch = 37

- **Weight Decay**
  We use weight decay = 0.0001

Results

Data	Accuracy	Macro F1	Micro F1
Training	87.09%	87.07%	94.04%
Validation	77.66%	77.59%	86.63%
Test	76.61%	76.52%	85.09%

- ## GN Variant

Normalization
This uses group normalization

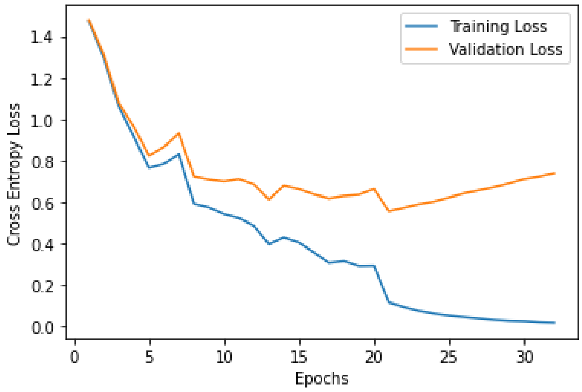
Hyperparameters

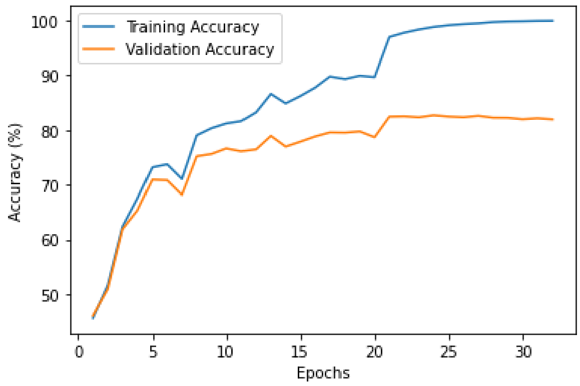
Optimizer	SGD
Learning Rate (initial)	0.1
Momentum	0.9
Weight Decay	0.0001
Batch Size	128
No of Group	16

After 20 epochs, the learning rate is changed to 0.01

Plots

The error and accuracy plots are obtained as shown:-





Regularization

- **Early stopping**
  Early stopping at epoch = 20

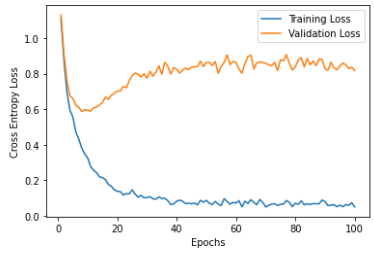
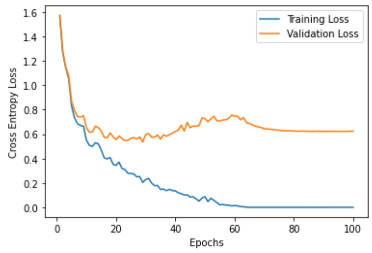
- **Weight Decay**
  We use weight decay = 0.0001

Results

Data	Accuracy	Macro F1	Micro F1
Training	96.93%	96.93%	98.79%
Validation	82.41%	82.41%	88.89%
Test	81.98%	82.00%	89.29%

1.2.4 Compare BN variant with 1.1 model

We take the model in 1.1 and add a batch norm layer (provided by pytorch). We compare it with our own implementation of batch norm

	Inbuilt Batch Norm	**Our Implementation**
Test Accuracy	79.41%	81.39%
Test Macro F1	79.50%	81.28%
Test Micro F1	88.17%	89.48%
Error Plot		

The results from our implementation are almost the same to the inbuilt batch norm provided by the pytorch. The error curves also converge to the same loss value.

1.2.5 Compare error curves and statistics of all the models

All the variants are compared below. Instance norm performs the best on all metrics for test data. The NN variant performs the worst. Within the normalization variants, LN performs the worst.

Accuracy

Variant	Training Data	Val. Data	Test Data
NN	85.78%	72.09%	71.74%
BN	93.19%	82.87%	81.39%
IN	**94.05%**	**83.99%**	**83.35%**
BIN	92.12%	83.31%	82.98%
LN	87.09%	77.66%	76.61%
GN	96.93%	82.41%	81.98%

Based, on Accuracy of test data, we can say
Performance : IN > BIN > GN > BN > LN > NN

Macro F1

Variant	Training Data	Val. Data	Test Data
NN	85.80%	72.04%	71.73%
BN	93.17%	82.78%	81.28%
IN	**94.08%**	**84.06%**	**83.43%**
BIN	92.14%	83.38%	83.00%
LN	87.07%	77.59%	76.52%
GN	96.93%	82.41%	82.00%

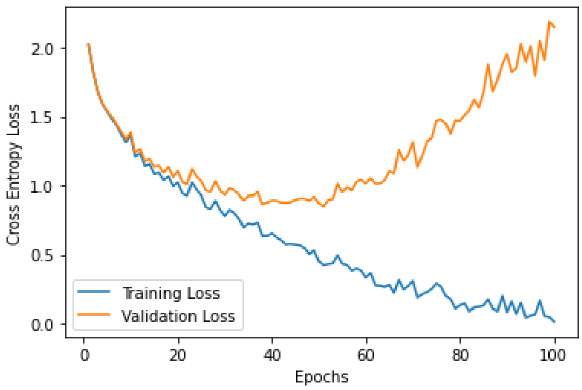
Based, on Macro F1 score of test data, we can say
Performance : IN > BIN > GN > BN > LN > NN

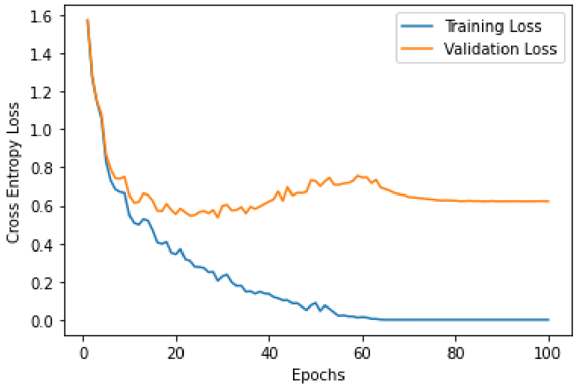
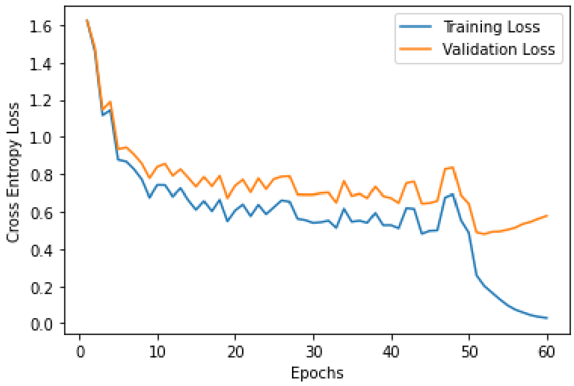
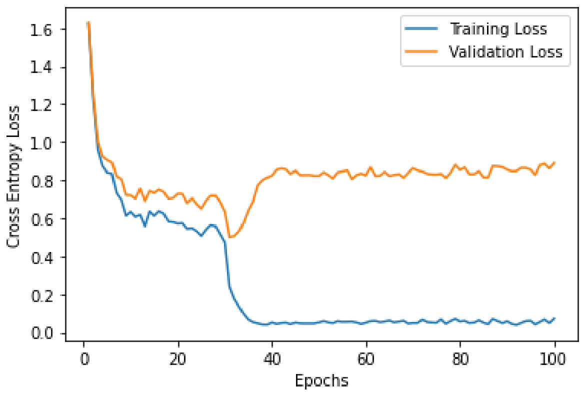
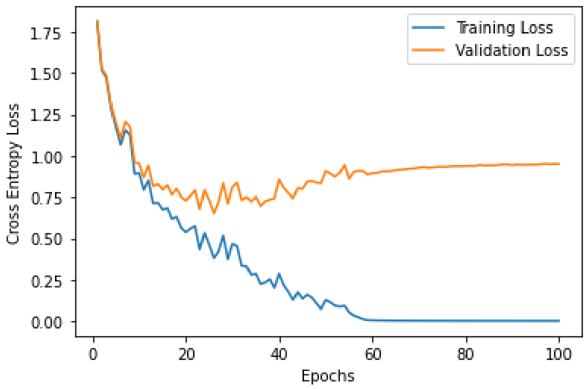
Micro F1

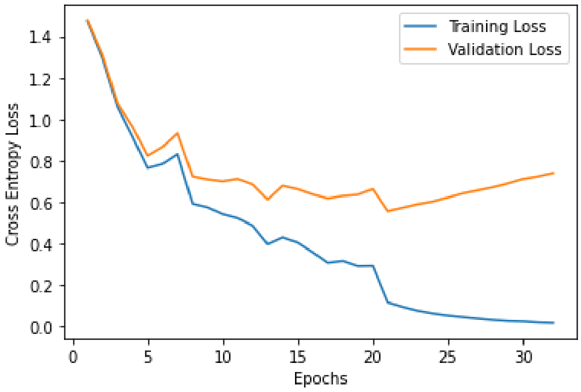
Variant	Training Data	Val. Data	Test Data
NN	93.52%	82.61%	82.26%
BN	96.48%	89.44%	89.48%
IN	**97.23%**	**90.19%**	**89.79%**
BIN	96.31%	90.32%	89.78%
LN	94.04%	86.63%	85.09%
GN	98.79%	88.89%	89.29%

Based, on Macro F1 score of test data, we can say
Performance : IN > BIN > BN > GN > LN > NN

Error Curves

Variant	Plot
NN	

BN	Training Loss Validation Loss Cross Entropy Loss Epochs
IN	Training Loss Validation Loss Cross Entropy Loss Epochs
BIN	Training Loss Validation Loss Cross Entropy Loss Epochs
LN	Training Loss Validation Loss Cross Entropy Loss Epochs

GN	Training Loss Validation Loss Cross Entropy Loss Epochs 1.4 1.2 1.0 0.8 0.6 0.4 0.2 0.0 0 5 10 15 20 25 30

1.2.6 Impact of Batch Size

We retrained the BN and GN variants for batch size 8. We do not observe any significant change of performance in any of the two variants. The performance almost remains the same. The effect might be more prominent if we take further small batch size. We get the following statistics:

Batch Norm

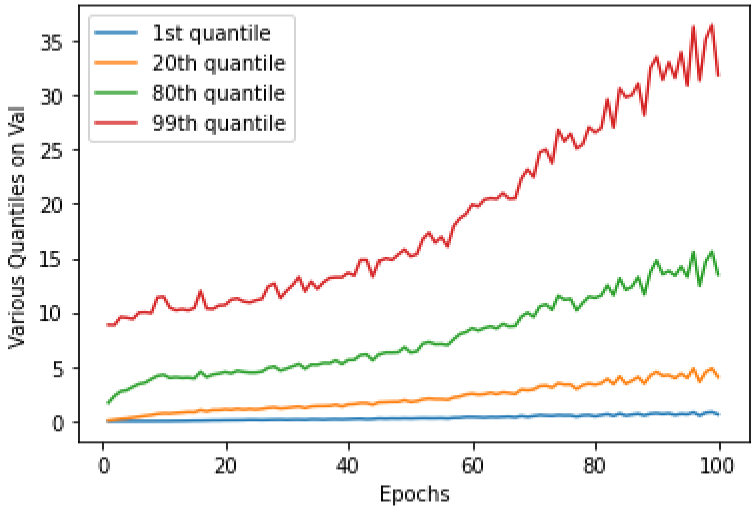
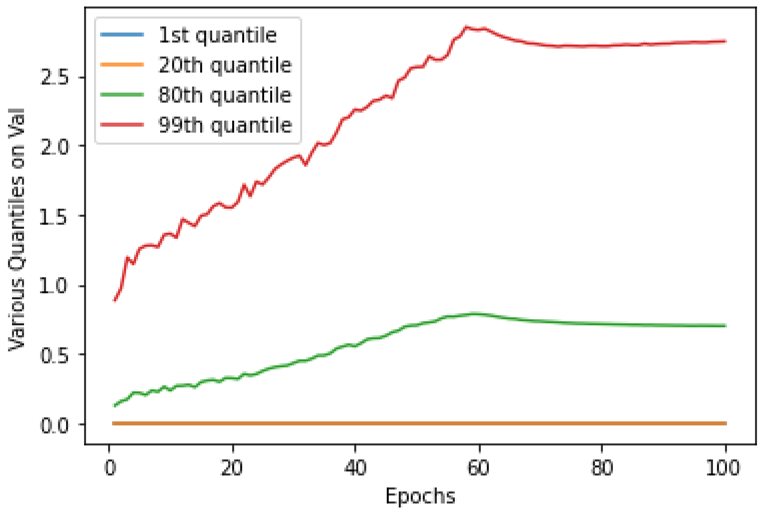
Data	Batch Size = 128			Batch Size = 8		
	Acc	Macro F1	Micro F1	Acc	Macro F1	Micro F1
Training	93.19%	93.17%	96.48%	92.78%	92.79%	96.79%
Val	82.87%	82.78%	89.44%	82.42%	82.38%	90.18%
Test	81.39%	81.28%	89.48%	81.39%	81.42%	88.99%

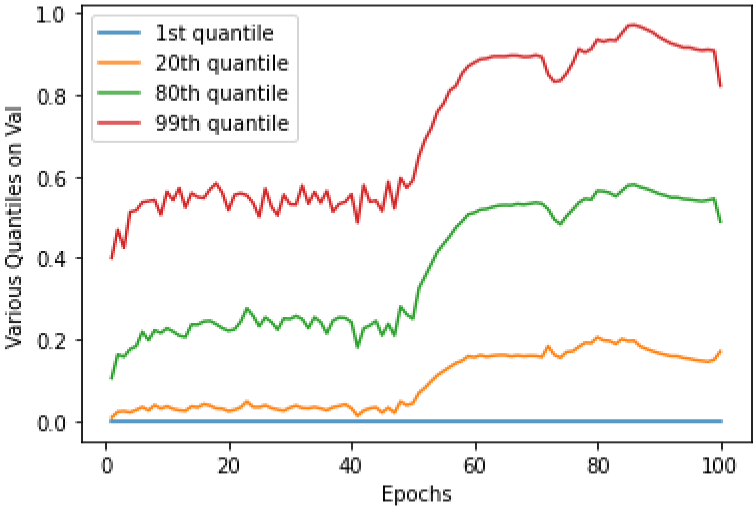
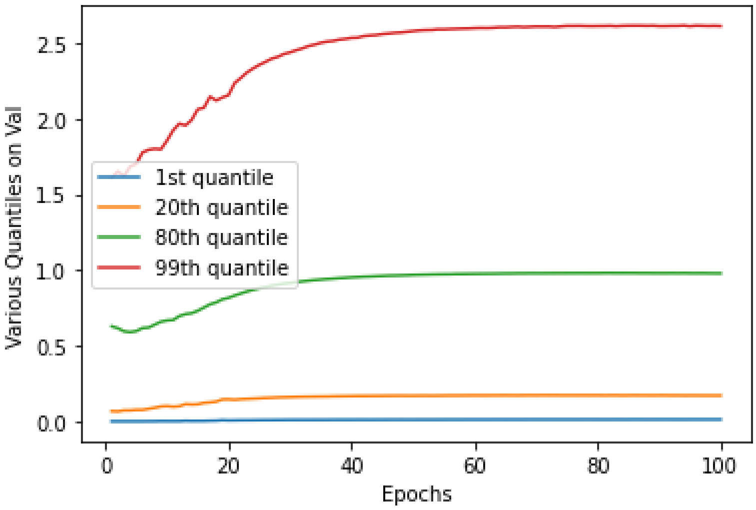
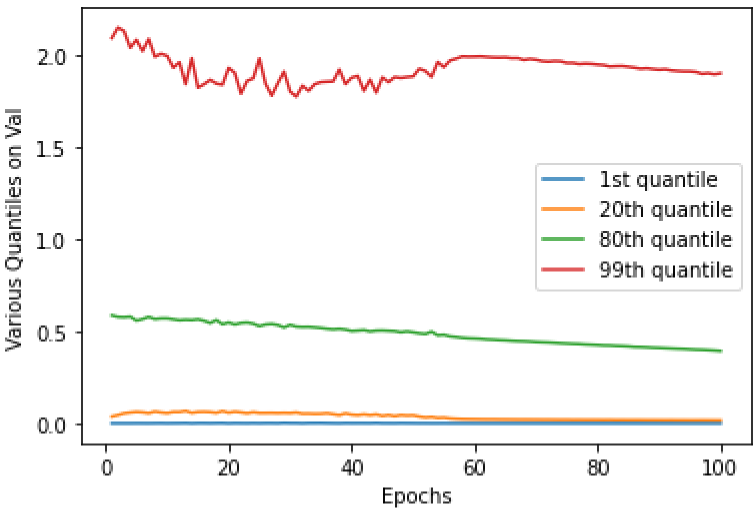
Group Norm

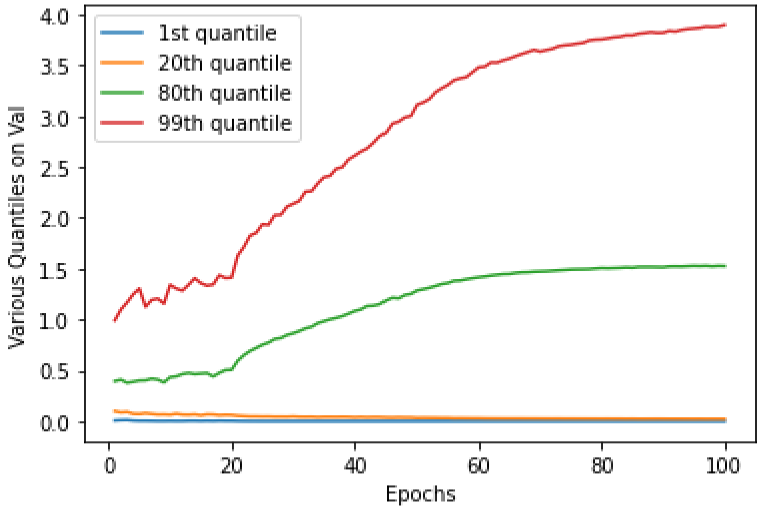
Data	Batch Size = 128			Batch Size = 8		
	Acc	Macro F1	Micro F1	Acc	Macro F1	Micro F1
Training	96.93%	96.93%	98.79%	89.05%	88.98%	94.88%
Val	82.41%	82.41%	88.89%	81.67%	81.57%	89.03%
Test	81.98%	82.00%	89.29%	81.26%	81.11%	88.78%

1.2.7 Evolution of feature distributions

We observe that in the later epochs the quantiles saturate for all the variants. It shows that after some epochs the input distribution that our model sees, becomes uniform. In other words, the learning becomes stable.

Variant	Plots
NN	
BN	

IN	1st quantile 20th quantile 80th quantile 99th quantile Various Quantiles on Val Epochs
BIN	1st quantile 20th quantile 80th quantile 99th quantile Various Quantiles on Val Epochs
LN	1st quantile 20th quantile 80th quantile 99th quantile Various Quantiles on Val Epochs

GN	 1st quantile 20th quantile 80th quantile 99th quantile Various Quantiles on Val Epochs 0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 0 20 40 60 80 100

Part 2:
LSTM with Layer Normalization and CRF

2.1 NER Tagging with Bi-LSTM

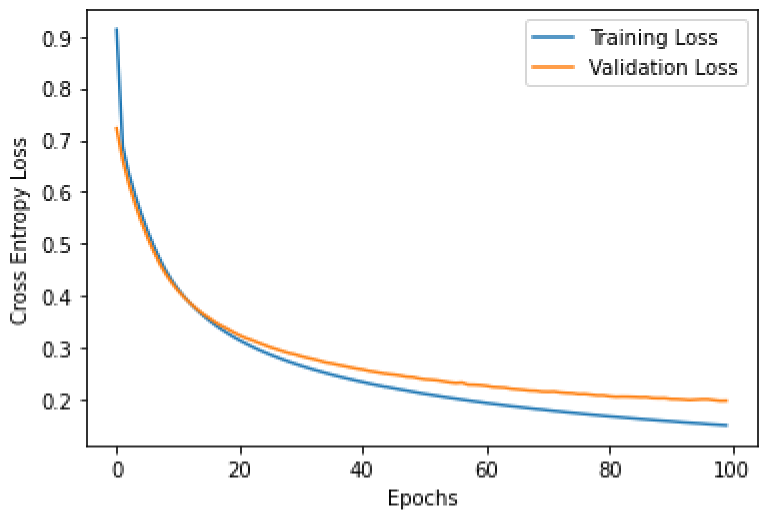
2.1.1. Bi-LSTM without pre-trained embeddings

In this part, we train a Bi-LSTM with random embeddings of size 100 that are trained from scratch at time of training.

Hyperparameters

Optimizer	SGD
Learning Rate	0.01
Momentum	0.9
Weight Decay	0.0001
Gradient Clipping	5 (p =2)
Batch Size	128

Plots



Results

For this part, we calculated the per class F1-score, Precision and Recall (**at instance level)** using the package seqeval. We also report the accuracy, which was evaluated at **token level**.

	Train Data			Val Data			Test Data		
	precision	recall	f1-score	precision	recall	f1-score	precision	recall	f1-score
art	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
eve	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
geo	0.75	0.76	0.76	0.71	0.75	0.73	0.71	0.75	0.73
gpe	0.84	0.81	0.82	0.84	0.79	0.81	0.83	0.81	0.82
nat	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
org	0.49	0.44	0.47	0.46	0.44	0.45	0.46	0.44	0.45
per	0.57	0.53	0.55	0.56	0.49	0.52	0.57	0.50	0.53
tim	0.79	0.74	0.76	0.79	0.73	0.76	0.77	0.72	0.74
micro avg	0.70	0.66	0.68	0.68	0.65	0.66	0.68	0.65	0.66
macro avg	0.43	0.41	0.42	0.42	0.40	0.41	0.42	0.40	0.41
weighted avg	0.69	0.66	0.68	0.67	0.65	0.66	0.67	0.65	0.66
	Acc: 0.95			**Acc: 0.94**			**Acc: 0.94**		

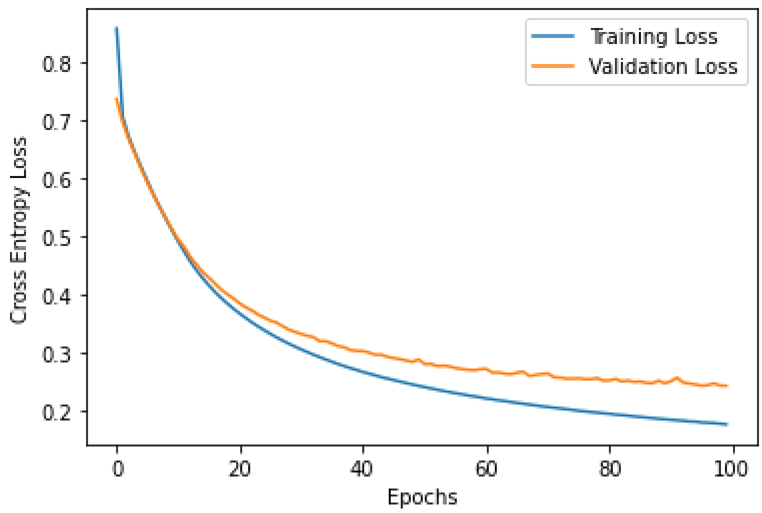
2.1.2. Bi-LSTM with pre-trained embeddings: Glove embeddings

In this part, we train a Bi-LSTM with the embeddings for our data vocabulary coming from the glove embeddings. The embeddings were then fine tuned while training for our dataset.

Hyperparameters

Optimizer	SGD
Learning Rate	0.01
Momentum	0.9
Weight Decay	0.0001
Gradient Clipping	5 (p =2)
Batch Size	128

Plots



Results

We calculated the per class F1-score, Precision and Recall (**at instance level)** using the package seqeval. We also report the accuracy, which was evaluated at **token level**.

Train Data	Val Data	Test Data

	precision	recall	f1-score
art	0.00	0.00	0.00
eve	0.00	0.00	0.00
geo	0.78	0.85	0.81
gpe	0.90	0.86	0.88
nat	0.00	0.00	0.00
org	0.56	0.48	0.52
per	0.70	0.73	0.72
tim	0.84	0.81	0.82
micro avg	0.76	0.75	0.76
macro avg	0.47	0.47	0.47
weighted avg	0.75	0.75	0.75

Acc: 0.96

	precision	recall	f1-score
art	0.00	0.00	0.00
eve	0.00	0.00	0.00
geo	0.77	0.83	0.80
gpe	0.90	0.86	0.88
nat	0.00	0.00	0.00
org	0.54	0.46	0.50
per	0.67	0.69	0.68
tim	0.83	0.80	0.82
micro avg	0.75	0.73	0.74
macro avg	0.47	0.45	0.46
weighted avg	0.74	0.73	0.74

Acc: 0.96

	precision	recall	f1-score
art	0.00	0.00	0.00
eve	1.00	0.01	0.02
geo	0.78	0.83	0.80
gpe	0.89	0.85	0.87
nat	0.00	0.00	0.00
org	0.53	0.45	0.49
per	0.69	0.70	0.70
tim	0.83	0.79	0.81
micro avg	0.75	0.74	0.74
macro avg	0.59	0.46	0.46
weighted avg	0.74	0.74	0.74

Acc: 0.96

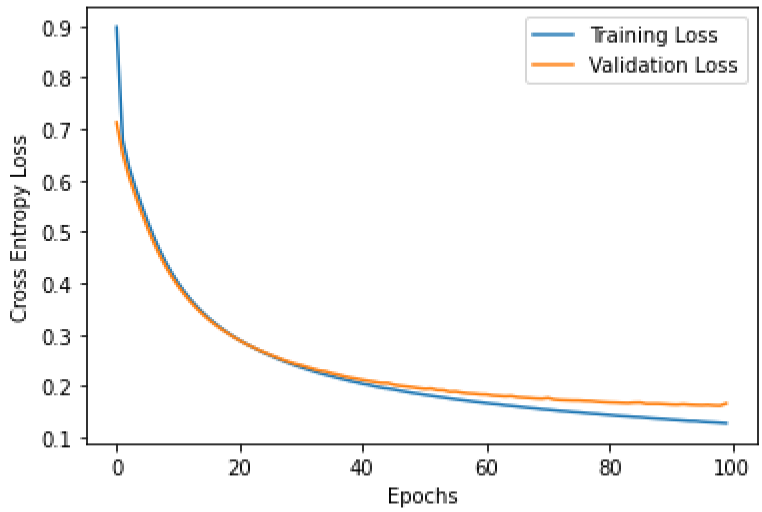
2.1.3. Bi-LSTM with Character Level Embeddings

In this part, we train a Bi-LSTM with the the embeddings for our data vocabulary coming from the glove embeddings plus character level embeddings coming from an additional Bi-LSTM trained at a alphabet level as compared to word level. Then the glove and character embeddings were combined and fed into the Bi-LSTM for training.

Hyperparameters

Optimizer	SGD
Learning Rate	0.01
Momentum	0.9
Weight Decay	0.0001
Gradient Clipping	5 (p =2)
Batch Size	128

Plots



Results

We calculated the per class F1-score, Precision and Recall (**at instance level)** using the package seqeval. We also report the accuracy, which was evaluated at **token level**.

Train Data

	precision	recall	f1-score
art	0.00	0.00	0.00
eve	0.00	0.00	0.00
geo	0.76	0.76	0.76
gpe	0.81	0.81	0.81
nat	0.00	0.00	0.00
org	0.49	0.44	0.46
per	0.58	0.56	0.57
tim	0.81	0.77	0.79
micro avg	0.70	0.67	0.69
macro avg	0.43	0.42	0.42
weighted avg	0.70	0.67	0.68

Acc: 0.95

Val Data

	precision	recall	f1-score
art	0.00	0.00	0.00
eve	0.00	0.00	0.00
geo	0.76	0.74	0.75
gpe	0.82	0.80	0.81
nat	0.00	0.00	0.00
org	0.48	0.42	0.45
per	0.54	0.52	0.53
tim	0.81	0.76	0.78
micro avg	0.70	0.66	0.68
macro avg	0.43	0.41	0.42
weighted avg	0.69	0.66	0.67

Acc: 0.94

Test Data

	precision	recall	f1-score
art	0.00	0.00	0.00
eve	0.00	0.00	0.00
geo	0.76	0.73	0.74
gpe	0.80	0.81	0.81
nat	0.00	0.00	0.00
org	0.48	0.43	0.45
per	0.57	0.54	0.56
tim	0.80	0.75	0.78
micro avg	0.70	0.66	0.68
macro avg	0.43	0.41	0.42
weighted avg	0.69	0.66	0.67

Acc: 0.94

2.1.4. Bi-LSTM with Layer Normalisation

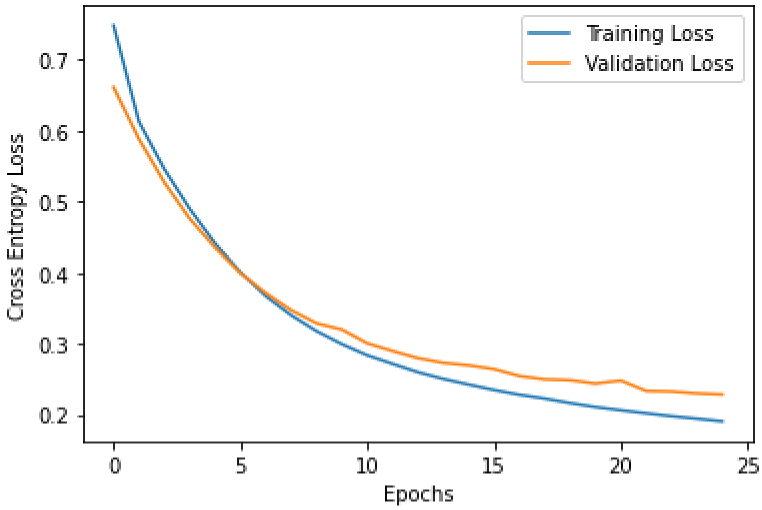
In this part, we train a Bi-LSTM with the the embeddings for our data vocabulary coming from the glove embeddings plus character level embeddings coming from an additional Bi-LSTM trained at a alphabet level as compared to word leve

Further, inside the nn.LSTM cell, we implement layer normalisation before passing the output h^t out of the LSTM cell.

Hyperparameters

Optimizer	SGD
Learning Rate	0.01
Momentum	0.9
Weight Decay	0.0001
Gradient Clipping	5 (p =2)
Batch Size	128

Plots



Results

We calculated the per class F1-score, Precision and Recall (**at instance level)** using the package seqeval. We also report the accuracy, which was evaluated at **token level**.

Train Data	Val Data	Test Data
Acc: 0.96	Acc: 0.96	Acc: 0.96

Train Data

_warn_prf(average, modifier, msg_start, len

	precision	recall	f1-score
art	0.00	0.00	0.00
eve	0.83	0.08	0.15
geo	0.80	0.87	0.83
gpe	0.92	0.87	0.90
nat	0.00	0.00	0.00
org	0.59	0.52	0.55
per	0.72	0.75	0.74
tim	0.85	0.81	0.83
micro avg	0.78	0.77	0.77
macro avg	0.59	0.49	0.50
weighted avg	0.77	0.77	0.77

Acc: 0.96

Val Data

	precision	recall	f1-score
art	0.00	0.00	0.00
eve	0.75	0.04	0.07
geo	0.78	0.84	0.81
gpe	0.92	0.86	0.89
nat	0.00	0.00	0.00
org	0.55	0.49	0.52
per	0.69	0.69	0.69
tim	0.84	0.80	0.82
micro avg	0.76	0.74	0.75
macro avg	0.57	0.46	0.47
weighted avg	0.75	0.74	0.75

Acc: 0.96

Test Data

	precision	recall	f1-score
art	0.00	0.00	0.00
eve	0.71	0.06	0.11
geo	0.79	0.85	0.81
gpe	0.91	0.86	0.89
nat	0.00	0.00	0.00
org	0.55	0.49	0.52
per	0.71	0.71	0.71
tim	0.84	0.79	0.81
micro avg	0.76	0.75	0.75
macro avg	0.56	0.47	0.48
weighted avg	0.75	0.75	0.75

Acc: 0.96

Comparison of Character Level Model to Pre-trained Embeddings

	Char Model: Macro F1	Glove Model: Macro F1
Train Data	0.68	0.75
Val Data	0.67	0.74
Test Data	0.67	0.74

We see that the glove model performs better as compared to the character level model. Firstly, the metric of comparison chosen to be is weighted Macro F1. As the data we have is imbalanced, looking at F1 will be unjust as there are few label classes like eve which has just 226 instances out of 86644 total instances. Because of such low data, no model is able to learn well on them, including the pre-trained ones.

Theoretically, Glove and Character level embedding both take the context into picture. Glove has embeddings based on the usage of words and captures a whole range of Language, only a part of which may be present in our model training data. Character level model is trained from scratch and we hope to learn morphological context by using them which our model does learn as a slight improvement over a normal random model. Still as the vocab we have in training data is low, we don't see large improvement.

Comparison of Normalised Model to Non-Normalised

	Norm Model: Macro F1	Non-Norm Model: Macro F1
Train Data	0.77	0.75
Val Data	0.76	0.74
Test Data	0.76	0.74

From the table above, we see that normalisation helps in increasing the Macro F1 score. As normalisation helps in stable learning by keeping the distribution of input constant, we see the improvement in the predictions. Normalisation even does good on the label classes which have a small number of samples. Thus, it makes the model better in a complete sense.

2.2 Linear chain CRF

In this part, we implemented a Linear Chain CRF from scratch. The partition function is used in calculating loss and training. The model does not directly give the predictions, it gives pre-softmax weights which are then used along with the transition matrix to calculate loss. Now we iterate through all possible outcomes (prediction) and choose the one with minimum loss. The sequence with minimum loss is called Most Likely Sequence.

Initialization: The transition matrix is initialised to zero in start. It is because a random initialisation will mean that we bias few labels to occur together and that may be wrong in the sense that it may not be true in real dataset. Such random initialisation will cause the model to go the wrong direction.

Zero initialisation means that we use start with no prior knowledge and learn the tendencies of labels to occur together directly from the dataset we have.

Results

	CRF Training Statistics			
Train	precision	recall	f1-score	support
art	0.00	0.00	0.00	296
eve	0.83	0.11	0.20	226
geo	0.75	0.79	0.77	29240
gpe	0.91	0.73	0.81	12058
nat	0.00	0.00	0.00	133
org	0.57	0.46	0.51	15803
per	0.71	0.54	0.61	13121
tim	0.86	0.76	0.81	15767
micro avg	0.76	0.67	0.71	86644
macro avg	0.58	0.42	0.46	86644
weighted avg	0.75	0.67	0.71	86644

Val	<pre> precision recall f1-score support
	
	         art 0.00 0.00 0.00 105
	         eve 0.73 0.10 0.18 78
	         geo 0.75 0.75 0.75 9724
	         gpe 0.91 0.72 0.80 4210
	         nat 0.00 0.00 0.00 50
	         org 0.55 0.43 0.48 5187
	         per 0.68 0.52 0.59 4457
	         tim 0.85 0.74 0.80 5254
	
	   micro avg 0.75 0.65 0.69 29065
	   macro avg 0.56 0.41 0.45 29065
	weighted avg 0.74 0.65 0.69 29065</pre>

Test	<pre> precision recall f1-score support
	
	         art 0.00 0.00 0.00 102
	         eve 0.62 0.06 0.11 87
	         geo 0.75 0.76 0.76 9912
	         gpe 0.90 0.72 0.80 4168
	         nat 0.00 0.00 0.00 55
	         org 0.54 0.43 0.48 5205
	         per 0.69 0.51 0.59 4406
	         tim 0.85 0.73 0.79 5275
	
	   micro avg 0.75 0.65 0.69 29210
	   macro avg 0.54 0.40 0.44 29210
	weighted avg 0.74 0.65 0.69 29210</pre>

Comparison with BI-LSTM models

First thing to note here is that CRF is very computationally expensive. The partition function calculation uses dynamic programming and still has a time complexity of $O(NC^2)$ where N is the number of data points and C is the number of classes.

	CRF Model: Macro F1	Norm Model: Macro F1
Train Data	0.71	0.77
Val Data	0.69	0.76
Test Data	0.69	0.76

Because of limited computational resources, we were only able to train CRF for 5 epochs. To our surprise, CRF did fairly well even in just 5 epochs. The results are not drastically different. If given sufficient training, we may achieve better stats.