Deep Learning Practical 1
MLPs, CNNs and Backpropagation

Qiao Ren
University of Amsterdam
Master's in Artificial Intelligence
qiao.ren@student.uva.nl
UvAnetID: 11828668

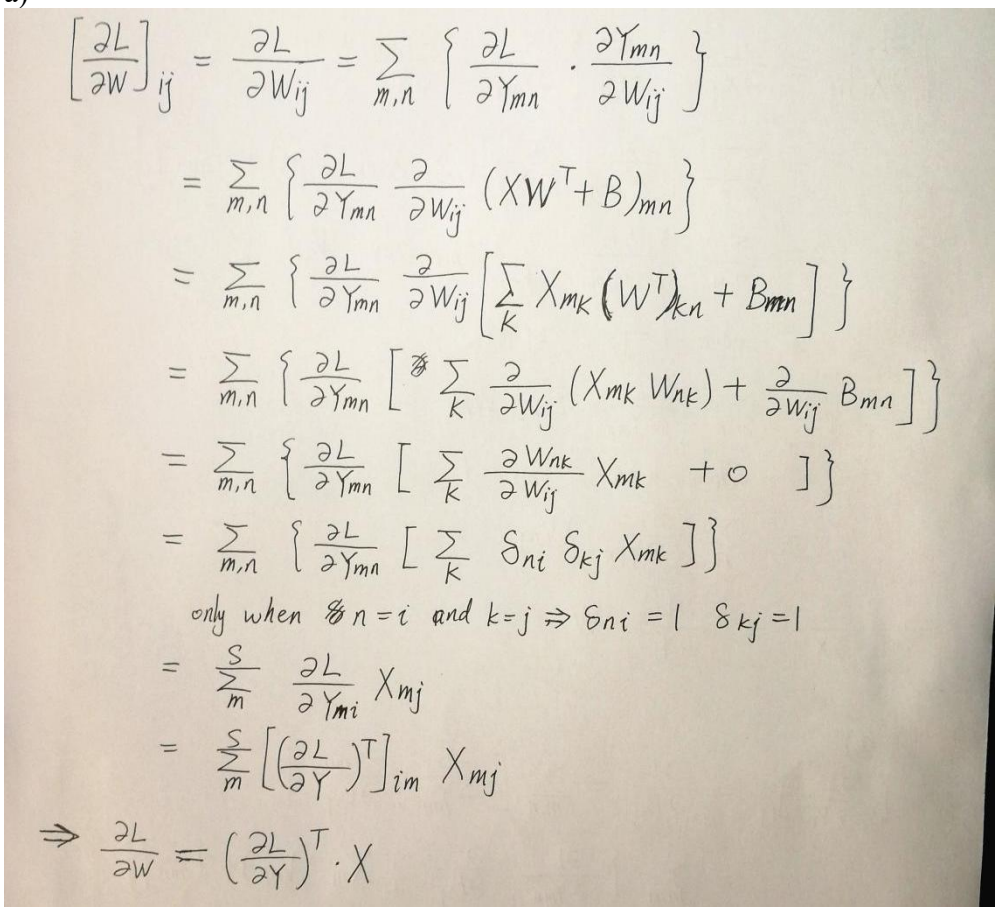
Abstract

This assignment covers multi-layer perceptron (MLP) and convolutional neural network (CNN).Gradient computation backpropagation is especially an important part in this assignment.

1 MLP backprop and numpy implementation

1.1 Evaluating the gradients

a)



1.1 a)

$$\left[\frac{\partial L}{\partial X}\right]_{ij} = \frac{\partial L}{\partial X_{ij}} = \sum_{m,n}\left\{\frac{\partial L}{\partial Y_{mn}} \cdot \frac{\partial Y_{mn}}{\partial X_{ij}}\right\}$$

$$= \sum_{m,n}\left\{\frac{\partial L}{\partial Y_{mn}} \frac{\partial}{\partial X_{ij}}\left[\left(\sum_k X_{mk} \cdot W_{nk}\right) + B_{mn}\right]\right\}$$

$$= \sum_{m,n}\left\{\frac{\partial L}{\partial Y_{mn}} \sum_k \delta_{mi}\,\delta_{kj} \cdot W_{nk}\right\}$$

when $k=j$, $\delta_{kj}=1$

$$= \sum_{m,n}\left\{\frac{\partial L}{\partial Y_{mn}}\,\delta_{mi}\,W_{nj}\right\}$$

when $m=i$, $\delta_{mi}=1$

$$= \sum_n \frac{\partial L}{\partial Y_{in}} W_{nj}$$

$$\Rightarrow \frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} \cdot W.$$

1.1 a)

$$\left[\frac{\partial L}{\partial B}\right]_{ij} = \frac{\partial L}{\partial B_{ij}} = \sum_{m,n}\left[\frac{\partial L}{\partial Y_{mn}} \cdot \frac{\partial Y_{mn}}{\partial B_{ij}}\right]$$

$$= \sum_{m,n}\left\{\frac{\partial L}{\partial Y_{mn}} \frac{\partial}{\partial B_{ij}}\left[\left(\sum_k X_{mk}\,W_{nk}\right) + B_{mn}\right]\right\}$$

$$= \sum_{m,n}\left\{\frac{\partial L}{\partial Y_{mn}} \cdot \left(0 + \frac{\partial B_{mn}}{\partial B_{ij}}\right)\right\}$$

$$= \sum_{m,n}\left\{\frac{\partial L}{\partial Y_{mn}} \cdot \delta_{mi}\,\delta_{nj}\right\}$$

when $m=i$ and $n=j \Rightarrow \delta_{mi}=1$ $\delta_{nj}=1$

$$= \frac{\partial L}{\partial Y_{ij}} = \left[\frac{\partial L}{\partial Y}\right]_{ij}$$

$$\frac{\partial L}{\partial B} = \left[\frac{\partial L}{\partial Y_{i1}} \quad \frac{\partial L}{\partial Y_{i2}} \quad \cdots \quad \frac{\partial L}{\partial Y_{in}}\right]$$

1.1 b)

$$\left[\frac{\partial L}{\partial X}\right]_{ij} = \frac{\partial L}{\partial X_{ij}} = \sum_{m,n}\frac{\partial L}{\partial Y_{mn}}\frac{\partial Y_{mn}}{\partial X_{ij}} = \sum_{m,n}\frac{\partial L}{\partial Y_{mn}}\frac{\partial}{\partial X_{ij}}\left[h(X_{mn})\right]$$

$$= \sum_{m,n}\frac{\partial L}{\partial Y_{mn}}\left[\frac{\partial h(X_{mn})}{\partial X_{ij}}\right]$$

$$\begin{cases}\text{when } m=i \text{ and } n=j \rightarrow \dfrac{\partial h(X_{mn})}{\partial X_{ij}} = \dfrac{\partial h(X_{ij})}{\partial X_{ij}} \\ \text{when } m\neq i \text{ or } n\neq j \rightarrow \dfrac{\partial h(X_{mn})}{\partial X_{ij}} = 0\end{cases}$$

$$= \frac{\partial L}{\partial Y_{ij}}\frac{\partial h(X_{ij})}{\partial X_{ij}}$$

$$\Rightarrow \frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} \circ \frac{\partial h(X)}{\partial X}$$

1.1 c) i) derivative of softmax

1.1 c) i) derivative of softmax.

$$Y_{ij} = [\text{softmax}(X)]_{ij} = \frac{e^{X_{ij}}}{\sum_k e^{X_{ik}}}$$

case ① when output index $ij \neq$ input index m, n

$$\frac{\partial Y_{ij}}{\partial X_{mn}} \overset{\text{quotient rule}}{=} \frac{\frac{\partial}{\partial X_{mn}} e^{X_{ij}} \cdot \sum_k e^{X_{ik}} - \left(\frac{\partial}{\partial X_{mn}} \sum_k e^{X_{ik}}\right) \cdot e^{X_{ij}}}{\left(\sum_k e^{X_{ik}}\right)^2}$$

(when $ik = m, n$)

$$= \frac{0 \cdot \sum_k e^{X_{ik}} - e^{X_{mn}} \cdot e^{X_{ij}}}{\left(\sum_k e^{X_{ik}}\right)^2}$$

$$= \frac{0 - e^{X_{mn}} \cdot e^{X_{ij}}}{\left(\sum_k e^{X_{ik}}\right)^2}$$

$$= -\frac{e^{X_{mn}}}{\left(\sum_k e^{X_{ik}}\right)} \cdot \frac{e^{X_{ij}}}{\sum_k e^{X_{ik}}}$$

$$= -Y_{mn} \cdot Y_{ij}$$

case ② when output index $i, j =$ input index m, n

$$\frac{\partial Y_{ij}}{\partial X_{mn}} \overset{\text{quotient rule}}{=} \frac{\left(\frac{\partial}{\partial X_{mn}} e^{X_{ij}}\right) \sum_k e^{X_{ik}} - \left(\frac{\partial}{\partial X_{mn}} \sum_k e^{X_{ik}}\right) \cdot e^{X_{ik}}}{\left(\sum_k e^{X_{ik}}\right)^2}$$

$$= \frac{e^{X_{mn}} \cdot \sum_k e^{X_{ik}} - e^{X_{mn}} \cdot e^{X_{ik}}}{\left(\sum_k e^{X_{ik}}\right)^2}$$

$$= \frac{e^{X_{mn}}}{\sum_k e^{X_{ik}}} \cdot \frac{\sum_k e^{X_{ik}} - e^{X_{ij}}}{\sum_k e^{X_{ik}}}$$

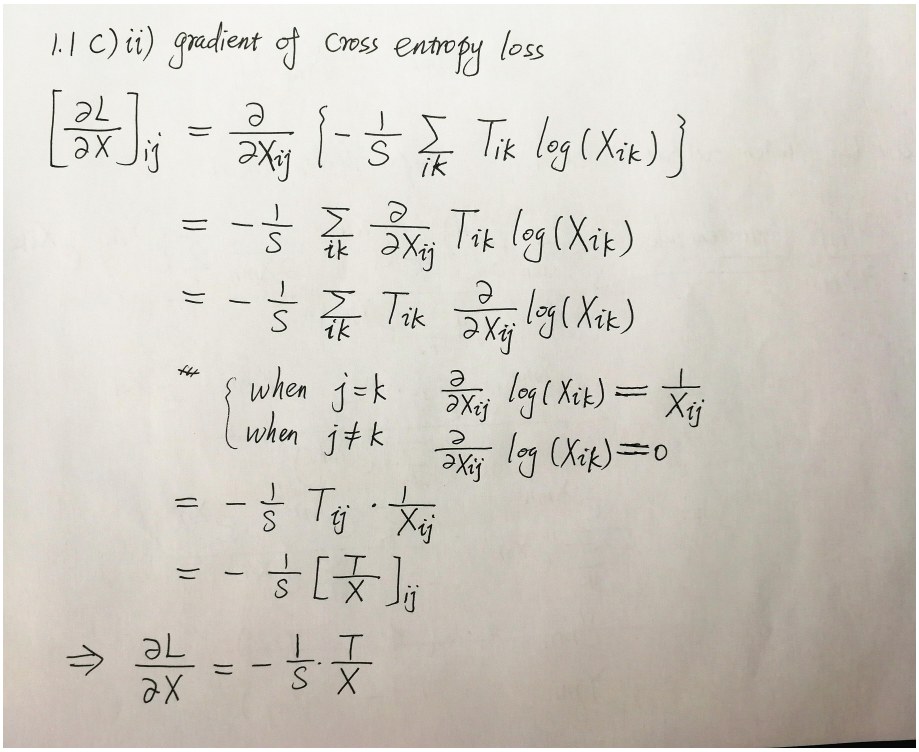
$$= y_{mn} \cdot (1 - y_{ij})$$

$$= Y_{mn} \cdot (1 - Y_{ij})$$

conclusion:

$$\left[\frac{\partial L}{\partial X}\right]_{mn} = \sum_{ij} \frac{\partial L}{\partial Y_{ij}} \frac{\partial Y_{ij}}{\partial X_{mn}} = \sum_{ij} \frac{\partial L}{\partial Y_{ij}} \cdot Y_{mn}(\delta_{mi}\delta_{nj} - Y_{ij})$$

$$= \begin{cases} \sum_{ij} \frac{\partial L}{\partial Y_{ij}} Y_{mn} \cdot (-Y_{ij}) & m, n \neq i, j \\ \sum_{ij} \frac{\partial L}{\partial Y_{ij}} Y_{mn}(1 - Y_{ij}) & m, n = i, j \end{cases}$$



1.2 **Numpy implementation**

With the default setting, the accuracy of the whole test set reaches 48.86%.

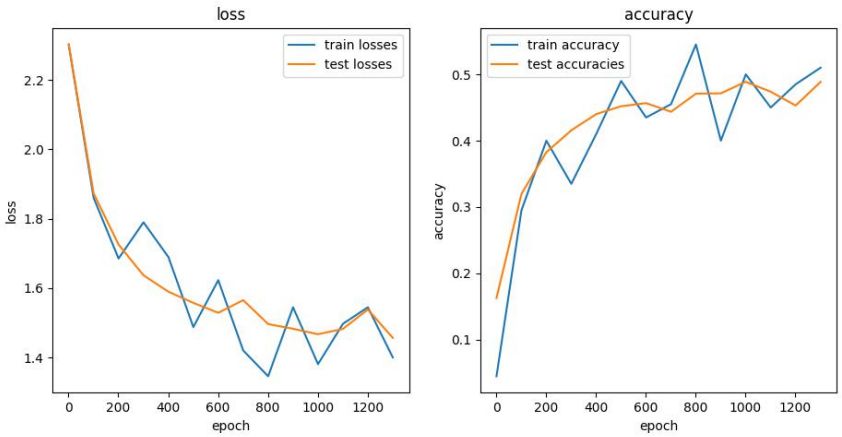


Image 1 MLP in numpy, default setting for parameters

2 Pytorch MLP

2.1

The loss and accuracy for default setting are in image 2.

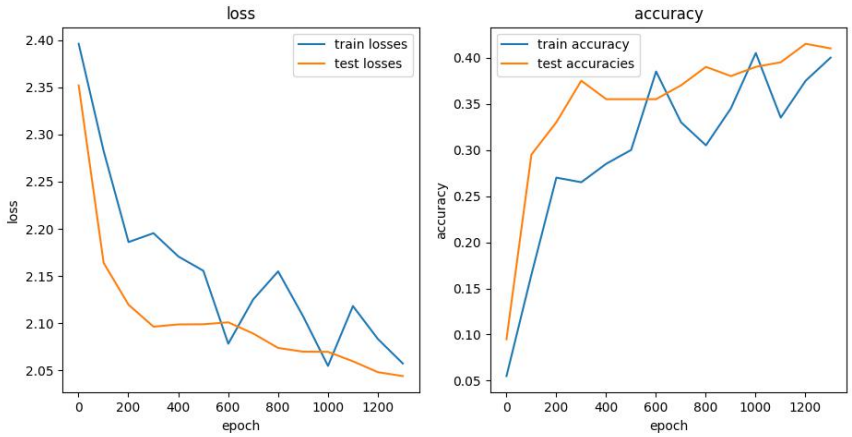


Image 2 MLP in pytorch, default setting for parameters

I made the following settings on the parameters. This optimized model largely improves the accuracy. Its accuracy on training set is 57%. Its accuracy on the test set is 55.5%. The loss and accuracy are in image 3.

- Batch normalization layers are added after each linear layer
- Hidden layer: [300, 500, 100]
- Optimizer: Adam
- Batch size: 200
- Max steps: 3000
- Learning rate: 0.0001

Justification on the parameter settings: Batch normalization is necessary because we need it to adjust the input distribution, in order to avoid extremely high or low values. It is important to build up more than one layers. I tried 1, 2 and 3 layers. It shows that 3 layers provides the highest accuracy. Because 1-layer and 2-layer networks are to simple to capture the complicated features in the data set. Too many layers would be harmful to the training. 3 layers is the best. Adam is better than SGD for cifar 10. Because Adam integrates momentum and correction bias. The learning rate is more adaptive to the gradient.

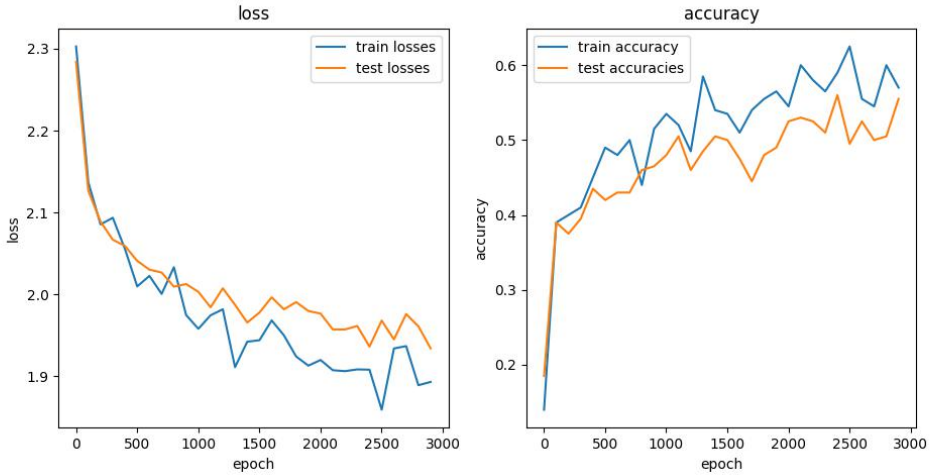


Image 3 MLP in pytorch, with optimized setting for parameters

2.2 Discuss about the benefits and/or the drawbacks of using the function *Tanh*

Benefit of Tanh: It has a strong gradient. Strong gradient is necessary for optimizing parameters. Its output are centered around 0. it is more efficient than sigmoid because it has a wider range for faster learning The distribution of its gradient values are consistent and stable among different layers.

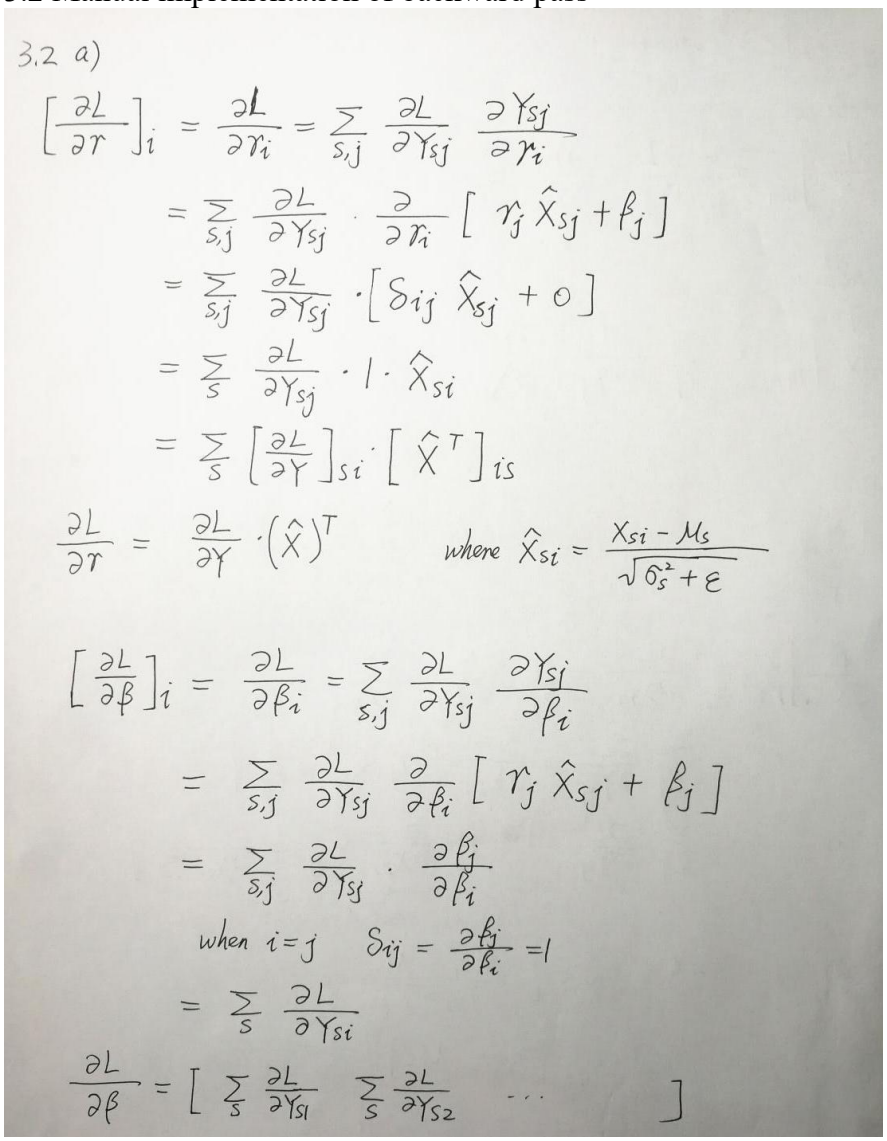
Drawback of Tanh: when x are very small x or very big, the Tanh output does not have big change. This means that the algorithm would hardly learn anything. This is causes a vanishing gradient problem. This can result in the network refusing to learn further, or being too slow to reach an accurate prediction.

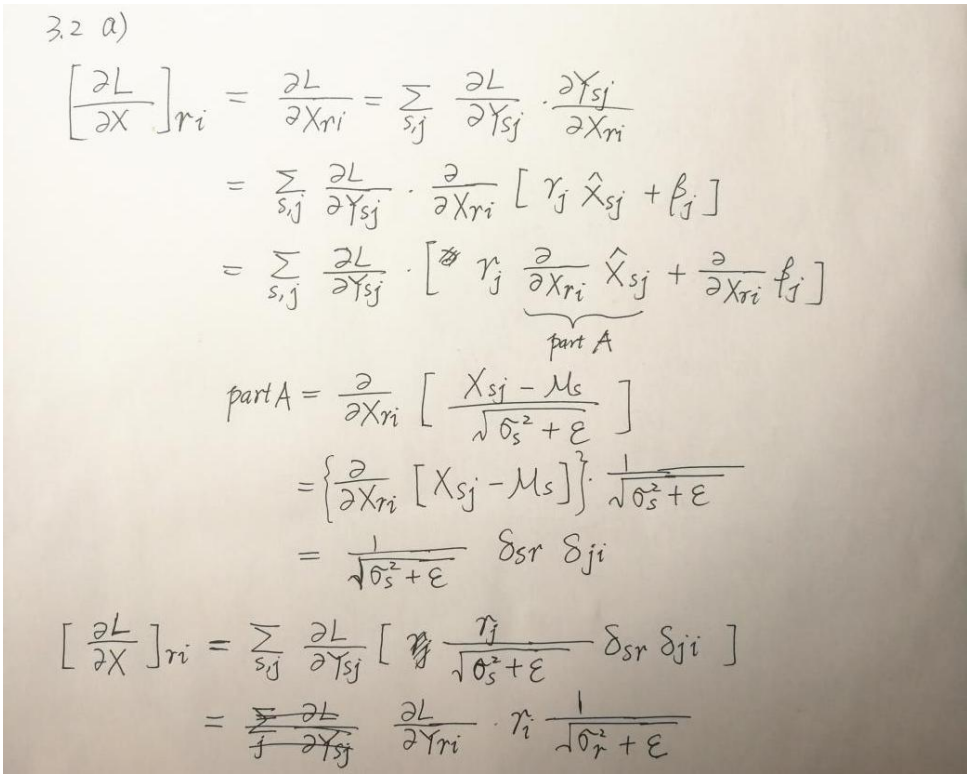
3 Custom module: layer normalization

3.1 Automatic differentiation

I have implemented it in the code.

3.2 Manual implementation of backward pass





3.2 d) compare the Layer Normalization approach with the Batch Normalization.

Batch normalization normalizes the input across the amount of data in a batch. In contrast, normalizes the inputs across the features.

Batch normalization: solves 2 problems.
1) Batch normalization limits covariate shift. It makes the activations to have zero mean and unit variance. It ensures each layer to learn on a more stable distribution of inputs. Therefore, it accelerates the training of the network.
2) Batch normalization restricts High order effect. High order effect is a problem that changing one weight has a big influence of other weights. Some weight parameters could be extremely large or extremely small. This problem forces us to use small learning rate. To deal with this problem, batch normalization restricts the weights from being imbalanced. Eventually, the training process is accelerated.

Limitation of batch normalization
1) batch normalization requires large amount of datapoints. Its computation cost is expensive. So batch normalization requires large amount of data for a mini batch.
2) batch normalization is not suitable for recurrent neural network. Because each time step requires a separate batch normalization layer. It is too complicated to do this.

Batch size has an influence on the batch normalization. If batch size =1, then variance =0 and batch normalization is not meaningful. So batch size should not be 1. If batch size are small, then the result could be noisy.

Layer normalization: Layer normalization sees inputs as independent. Each input has a different normalization operation. Any batch size can be used in layer normalization. Layer normalization has a good performance on RNN. Mean and variance of one data point are independent of other data points.

Batch size does not influence layer normalization. Because layer normalization goes over the features.

4 Pytorch CNN