Ao Zhang, 0300039680
Lingfeng Zhang, 0300134245

Assignment 2

1 Question 1

Since the question asks us to hand-derive back propagation of the defined neural network architecture, a forward graph is firstly built as Figure 1 for showing the relationships between all those nodes.

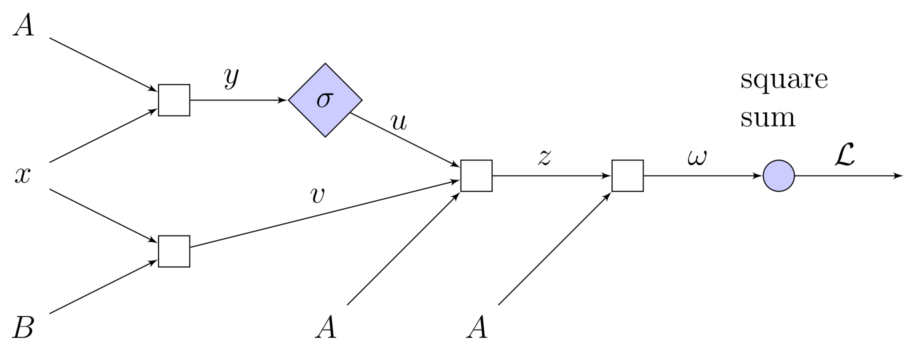


Figure 1: Forward graph $\mathcal{G}$.

Also, as mentioned in the lecture, the back propagation is the dual function of the forward gradient propagation. Therefore, in order to get the back propagation gradient, we firstly should derive the forward gradient graph, shown as Figure 2

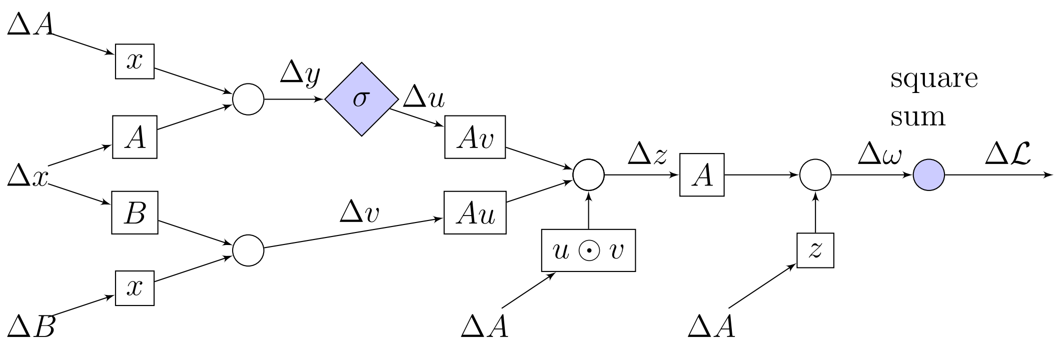


Figure 2: Forward Gradient graph $\partial\mathcal{G}$.

Now, we could easily get the back propagation graph by transposing forward graph, which is,

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \left(\partial \mathcal{G}\right)^T$$

Notice that, the parameter A appears 3 times in the graph. So according to the chain rule, the result of the back propagation gradients w.r.t each parameter are,

$$\frac{\partial \mathcal{L}}{\partial A} = \frac{\partial \mathcal{L}}{\partial \omega}\frac{\partial \omega}{\partial z} \odot \frac{\partial z}{\partial u} \odot \frac{\partial u}{\partial y}\frac{\partial y}{\partial A} + \frac{\partial \mathcal{L}}{\partial \omega}\frac{\partial \omega}{\partial z}\frac{\partial z}{\partial A} + \frac{\partial \mathcal{L}}{\partial \omega}\frac{\partial \omega}{\partial A}$$
$$\frac{\partial \mathcal{L}}{\partial B} = \frac{\partial \mathcal{L}}{\partial \omega}\frac{\partial \omega}{\partial z} \odot \frac{\partial z}{\partial v}\frac{\partial v}{\partial B}$$

where, the reason why there appears $\odot$ is that the third function $z = A(u \odot v)$ is the element-wise multiplication.

So, the upper equations are expanded to,

$$\frac{\partial \mathcal{L}}{\partial A} = 2\omega Z^T + 2A^T\omega(u \odot v)^T + \sigma(2A^TA^T\omega \odot v) \odot (1 - \sigma(2A^TA^T\omega \odot v))x^T$$
$$\frac{\partial \mathcal{L}}{\partial B} = (2A^TA^T\omega \odot u)x^T$$

After implementing in the python code, we produce a random input and the result of the loss function is shown as Figure 3

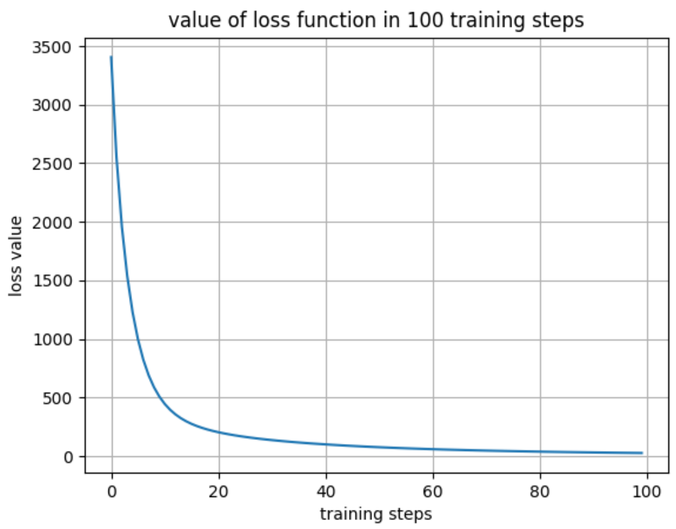


Figure 3: Results of back propagation.

The parameters settings are shown below,

According to the result, we could conclude the gradient based back propagation could make minimize the loss function and make the model learn.

Table 1: All relative parameters values.

Parameter	Value
N (number of inputs)	100
K (input dimensions)	5
optimization	Gradient Descent

2 Question 2

For softmax function in this question, K represents K kind of classes and m represents m kind of features for each input x. So $Wx \in \mathbb{R}^{\mathbb{K}\times\mathbb{1}}$ means there is a value for each class and softmax(Wx) gives each class with a probability. W is just a symbol, so I use another symbol $F \in \mathbb{R}^{\mathbb{K}\times\mathbb{K}}$ there. Softmax(Wx) and Softmax(Fx) are in the same hypothesis space, and they have same probability distribution because they can be trained and reached same result finally whether their initialization are same or not.

Due to their distribution are same, expectation can be used to proof this question.

There are two matrix expectation formula which should be introduced.

$$E(AX) = AE(X)$$
$$E(A+B) = E(A) + E(B)$$

In order to proving the problem, we introduce another 3 hypothesis,

$$D = AC,\ A \in \mathbb{R}^{K\times K}, C \in \mathbb{R}^{K\times K}$$
$$E = BC,\ B \in \mathbb{R}^{K\times K}, C \in \mathbb{R}^{K\times K}$$
$$F = D + E,\ D \in \mathbb{R}^{K\times K}, E \in \mathbb{R}^{K\times K}$$

where, since $A \in \mathbb{R}^{K\times K}, B \in \mathbb{R}^{K\times K}, C \in \mathbb{R}^{K\times K}$, according to the linear combination, $D \in \mathbb{R}^{K\times K}$. Similarly, $E \in \mathbb{R}^{K\times K}$ and $F \in \mathbb{R}^{K\times K}$.

From another point of view, since for each member hypothesis in $\mathcal{H}_1$ and $\mathcal{H}_2$ specifies a $p_{Y|X}$, the relationship of the expectation can be derived between D,E,F and A,B,C,

$$\begin{aligned}
E(Dx) &= E(ACx)\\
&= AE(Cx)\\
E(Ex) &= E(BCx)\\
&= BE(Cx)\\
E(Fx) &= E((D+E)x)\\
&= E(Dx+Ex)\\
&= E(Dx)+E(Ex)\\
&= AE(Cx)+BE(Cx)\\
&= (A+B)E(Cx)\\
&= E((A+B)Cx)
\end{aligned}$$

Due to Softmax(Wx) and Softmax(Fx) have the same distribiton, their expecation are same too.

$$E(Softmax(Wx)) = E(Softmax(Fx))$$

Softmax use the way of probability to represent expectation, and the way of values can also be represented by expectation,meaning the expectation of Wx and of Fx are same,too.So,

$$\begin{aligned}
E(Wx) &= E(Fx)\\
&= E((A+B)Cx)
\end{aligned}$$

Back to the probability way,

$$E(Softmax(Wx)) = E(Softmax((A+B)Cx))$$

Because Softmax(Wx) and Softmax((A+B)Cx) have same expectation, they have same probability distribution.

Therefore,

$$\mathcal{H}_1 = \mathcal{H}_2$$

3 Question 3

Since the question asks us to define the model first, we will introduce the model we used for the test first.

3.1 Model Introduction

Soft-max Regression

Simply, we define the soft-max regression as,

$$y = \mathbf{softmax}\Big(Wx + b\Big)$$

MLP

The MLP we defined is shown as Figure 4.

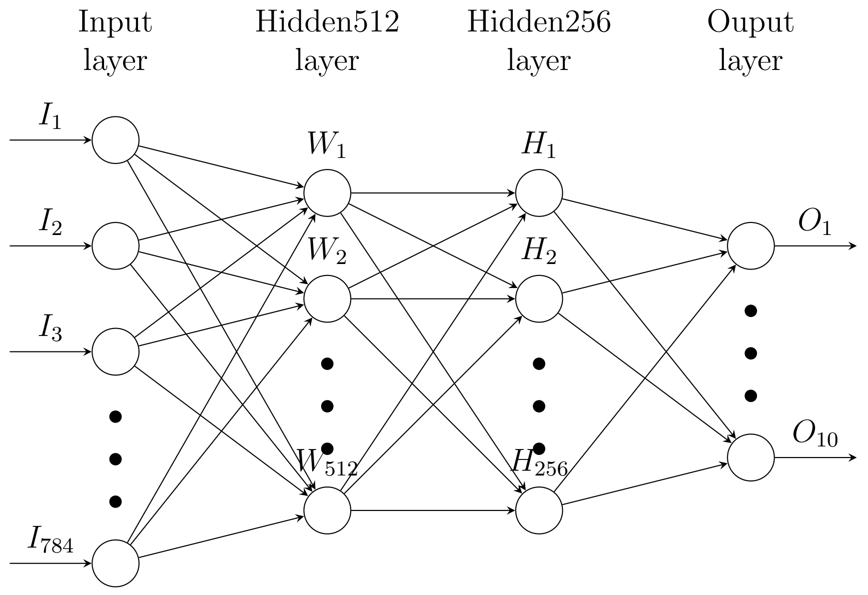


Figure 4: MLP model for the question 3.

We set two hidden layers, with first hidden layers having 512 nodes and second hidden layer having 256 nodes.

The mathematical model could be expressed as,

$$\hat{Y} = \sigma\Big(W_3 \cdot \text{relu}\Big(W_2 \cdot \text{relu}\big(W_1 X + b_1\big) + b_2\Big) + b_3\Big)$$

where,

$$X \in \mathbb{R}^{N\times 784},\ W_1 \in \mathbb{R}^{784\times 512},\ W_2 \in \mathbb{R}^{512\times 256},\ W_3 \in \mathbb{R}^{256\times 10} b_1 \in \mathbb{R}^{512},\ b_2 \in \mathbb{R}^{256},\ b_3 \in \mathbb{R}^{10}$$

CNN

The structure of CNN used in this assignment could be seen as Figure 5.

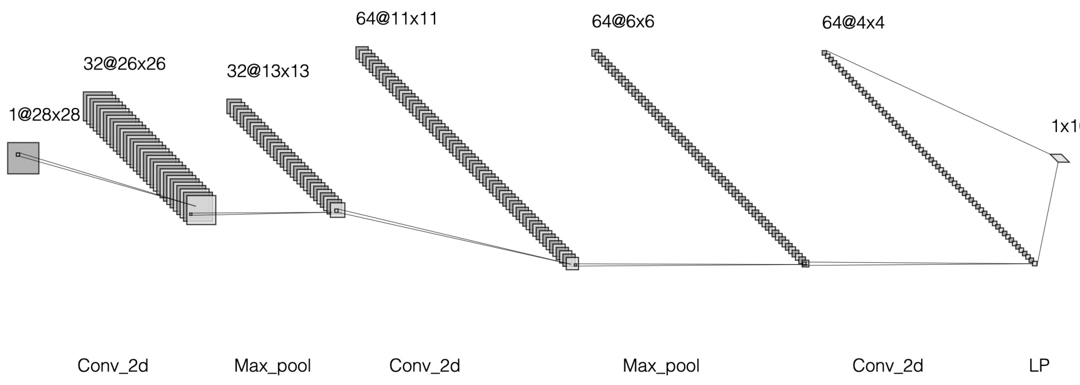


Figure 5: CNN model for the question 3.

Other parameters settings

Since we want to find out the influences of batch normalization and drop out on different models, we set all other parameters as the same during the whole training process. The parameter settings is shown below,

Table 2: All relative parameters values.

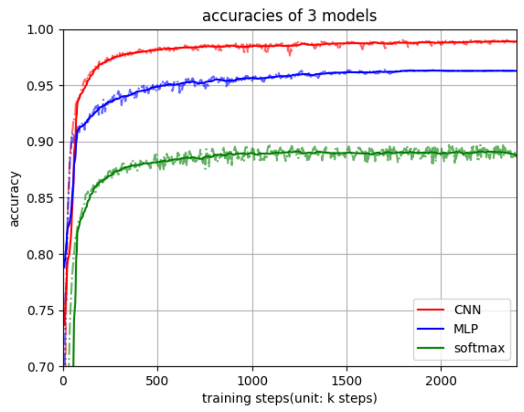
Parameter	Value
learning rate	0.001
batch size	500
epoches	20
dropout rate	0.5
learning rate decay coefficient	0.96
learning rate decay steps	100
optimization method	Adam
loss function	cross entropy

3.2 Experiments and Conclusion

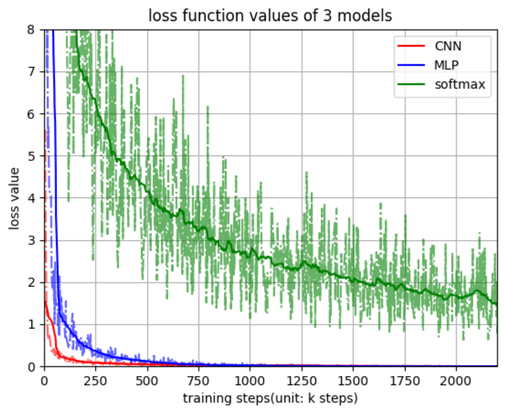
In the following section, we will first compare the results of all three models on the loss function values and the accuracies respectively. Then we will dive into the influence of batch normalization and dropout on each model separately. A brief conclusion will be drawn after each comparison. And hopefully, we could be able to cover the main features of each plot.

3.2.1 Comparison between 3 models

The results of the training loss values and validation accuracies are shown as Figure 6



(a) Accuracy of the validation set of each model.



(b) Training loss values of each model.

Figure 6: Results on training set and validation during the whole training process of 3 models.

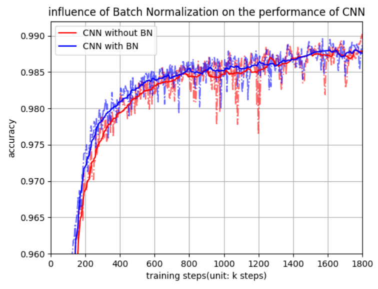
Some predominant characters could be observed as: The loss function of CNN and MLP converge extremely faster than soft-max regression; For the accuracy, CNN is greater than MLP. But they both dramatically outperform the soft-max regression. Several reasons for the phenomenon above could be concluded as,

- Since the parameters sizes of both MLP and CNN are significantly greater than that of soft-max regression, they could naturally outperform the soft-max regression model.
- Though the parameters sizes of MLP and CNN themselves differ slightly, the main reason why CNN could outperform MLP may not involve the parameters size. One possible reason is that the feature maps extracted from CNN may give more intuitive information about the differences between different classes. Thus, it could make the model predict better.

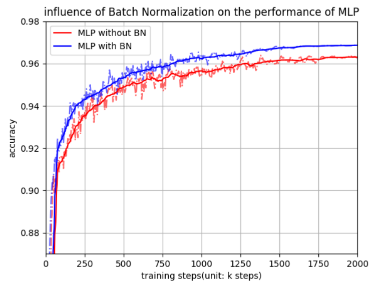
3.2.2 Influence of Batch Normalization

According to the result in the first section, we can easily find out that different models have different performances, and the scope of the plots are different too. Therefore, in order to show the difference of each model with and without batch normalization, we separate the plots according to the models.

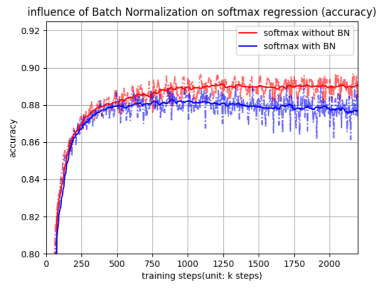
The results on the accuracies and loss values are shown as Figure 7.



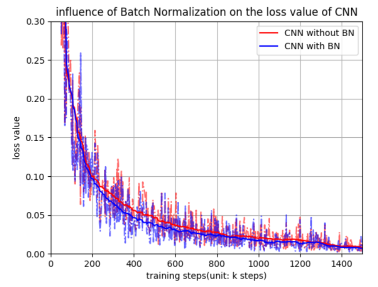
(a) Influence of BN on accuracy of CNN



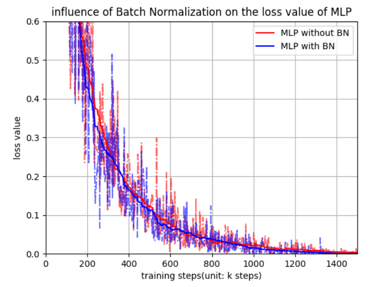
(b) Influence of BN on accuracy of MLP



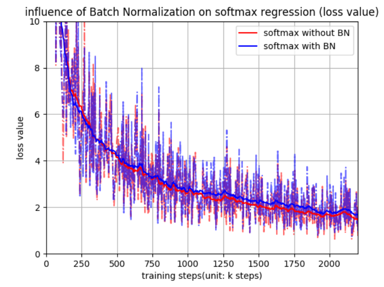
(c) Influence of BN on accuracy of soft-max



(d) Influence of BN on loss of CNN



(e) Influence of BN on loss of MLP



(f) Influence of BN on loss of soft-max

Figure 7: Influence of Batch Normalization on 3 models.

As the question requests, we will analyse the features of the graphs of each model individually in the following part.

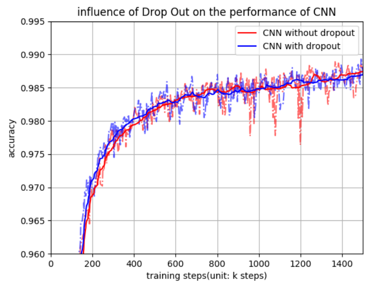
- **For CNN**: we can observe that the performance after batch normalization(BN) goes a little bit higher than before BN within 1200 training steps. Meanwhile, the loss value after BN also converges faster than before BN. After about 1200 steps, they both converge to the same level. Potential reasons for this could be seen as: since the BN could actually improve the ability of model to deal with covariance shift, it could help the training process to find the optimal solution faster. While, the training dataset for this specific task is large enough, which results the same performance when the training process goes further.

- **For MLP**: obviously, from testing accuracy, BN significantly improves the performance of the model, while the loss value during the whole training process remains the same. The result basically follows the theory we learnt from the class, that is the BN not only helps the model learn faster, but also helps increase the accuracy of the prediction.

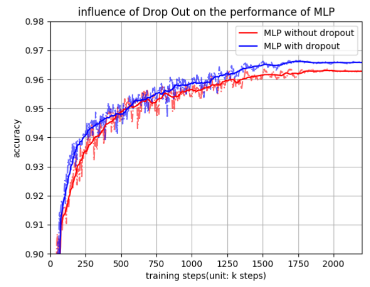
- **For soft-max**: though the loss function remains the same before and after BN, it is weird to see that the performance after BN is actually worse than before. I think one of the possible reasons is that, since the soft-max regression only has one layer before gating function, the actual prediction probability distribution would be changed when we apply BN on the model. It sabotages the true prediction distribution that the original model could possibly have. Therefore, it drives the accuracy away when after 500 training steps.

3.2.3 Influence of Dropout

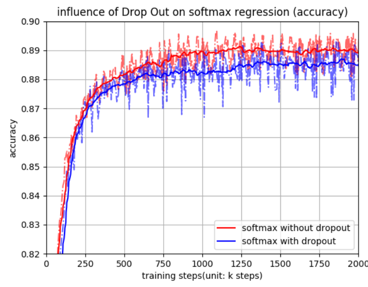
Also, in this section, we will draw the conclusion with respect to each model. All of the results is shown as Figure 8



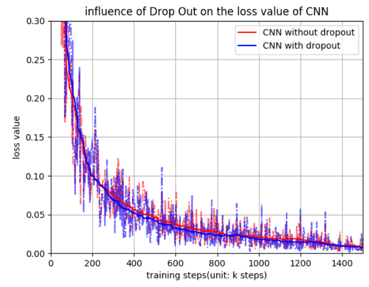
(a) Influence of dropout on accuracy of CNN



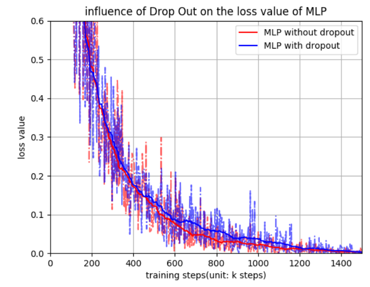
(b) Influence of dropout on accuracy of MLP



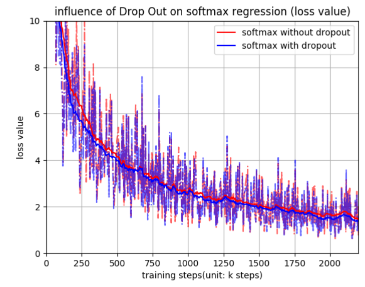
(c) Influence of dropout on accuracy of soft-max



(d) Influence of dropout on loss of CNN



(e) Influence of dropout on loss of MLP



(f) Influence of dropout on loss of soft-max

Figure 8: Influence of Dropout on 3 models.

The conclusions would be,

- **For CNN**: since the Convolutional Kernel itself could not be applied with dropout, the only way to do the dropout in the model is the dropout in the last layer(normal linear layer with soft-max gating function). Therefore, as we can see, the dropout itself does not influence the model at all. This basically agrees with our expectation.

- **For MLP**: dropout is used in every single layer of this model. As a result, the accuracy of the model after dropout shows promising improvement than before dropout. The loss function remains the same during the whole process. This is because the dropout, acting as a regularizer, effectively improve the ability of model to deal with the noise. This basically follows the theory we learnt from the class.

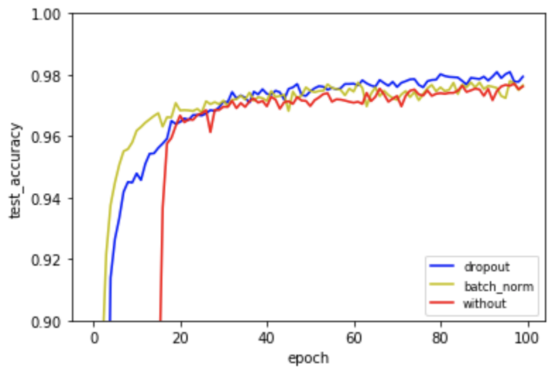
- **For soft-max**: apparently, dropout considerably damages the performance of soft-max regression model. The possible explanation is similar to the BN on soft-max. Since there is only one layer, if we apply dropout right before soft-max gating, it could change the probability distribution of the prediction, which is the culprit of worsening the model.

3.3 Understanding Batch Normalization with some experiments

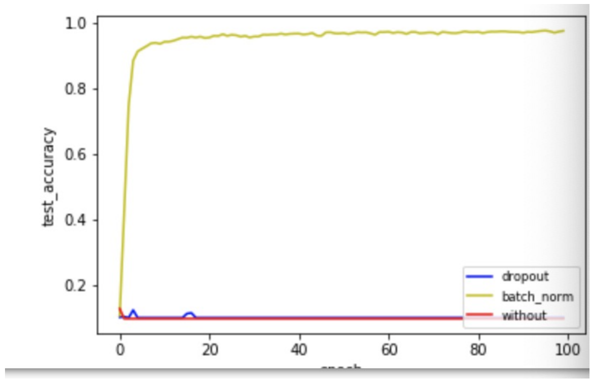
In this part, some interesting observation about batch normalization will be illustrated according to the experiments. This part meets the additional requirement of this question.

3.3.1 Gradient Vanishing

In the MLP model, the accuracy of MLP model with more hidden layers without batch normalization drops down extremely low sometimes. This is because gradient vanishing happens. Since batch norm can figure this issue out, the accuracy with batch normalization is still high when gradient vanishing occurs. As shown in Figure 9, MLP with shallow hidden layers(2 layers) performs well in three models with different conditions, but MLP with more hidden layers(5 layers) performs bad in these models except the model with batch normalization.



(a) MLP with 2 hidden layers



(b) MLP with 5 hidden layers

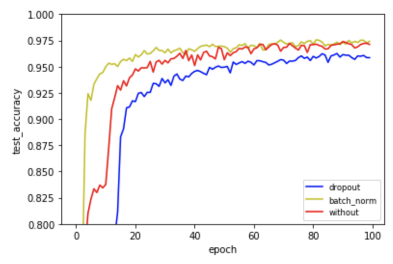
Figure 9: Gradient vanishing comparison

When the number of hidden layers increasing, the probability distribution is intend to be located in gradient saturation region,so the model,to be exact,the parameters are difficult to learn because their ”inputs” in hidden layer are 0s. Any parameters multiply to 0 is 0, so the parameters become meaningless.In these experiments, the activation function RELU is applied. If the probability distribution is constrained in negative side of X axis,which is covariate shift, when using RELU activation function, the distribution becomes 0, so gradient vanishing happens. However, the batch normalization can deal with the issue of covariate shift, and the distribution is hard to be constrained in gradient saturation region, so the model with batch normalization can perform well in more hidden layers MLP.

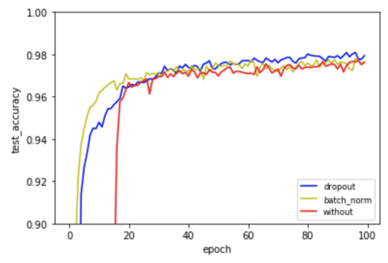
3.3.2 Some properties of batch normalization

As shown in Figure 10, 9 models with there different batch size(100,300,500) are compared to explain some interesting properties of batch normalization.

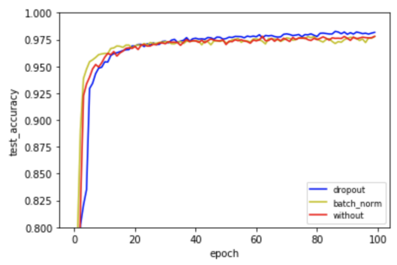
In 100 batch size, the accuracy of the model with dropout is lower than the accuracy without dropout. One possible reason is the model with 100 batch size is a little bit underfitting. Small batch size model has the similar effect of dropout because small batch size data are more noisy, so enlarging batch size from 100 to 500 makes the models become overfitting possible. It works because dropout makes effort to improve the accuracy. On the other hand, due to the batch size become larger, the regularization effect of batch normalization become less. Batch normalization has at least two properties, one of which is faster learning, this illustrates why the model with batch normalization can converge faster(close to y axis). Another property is a slight effect of regularization.Small batch size has larger effect of regularization w.r.t batch normalization because small batch size data are more noisy and vice versa, large batch size has less effect of regularization w.r.t batch normalization because large batch size data are less noisy.This is the reason why the accuracy of batch normalization is obvious larger than that without batch normalization in small batch size(batch size = 100) and the accuracy of batch normalization is similar as that without batch normalization in large batch size(batch size = 500) in these plots. To trade-off these two issues, batch size 300 is introduced. the accuracy of the models with batch normalization or dropout are both slightly larger than the third model.



(a) batch size = 100



(b) batch size = 300



(c) batch size = 500

Figure 10: 2 hidden layers MLP with different batch size