Yue Zhou
29 March 2020

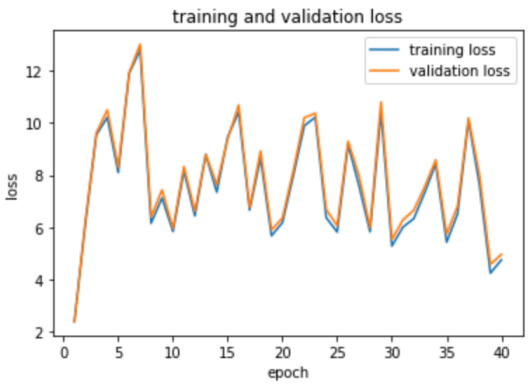
DD2424 Report assignment 1-bonus

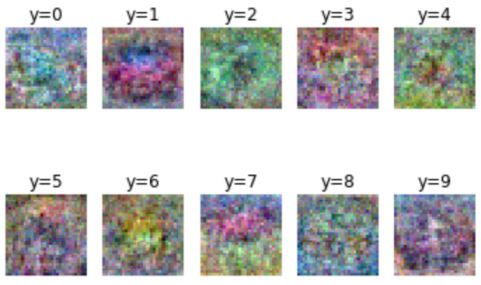
Yue Zhou

1. Optimize the performance of the network

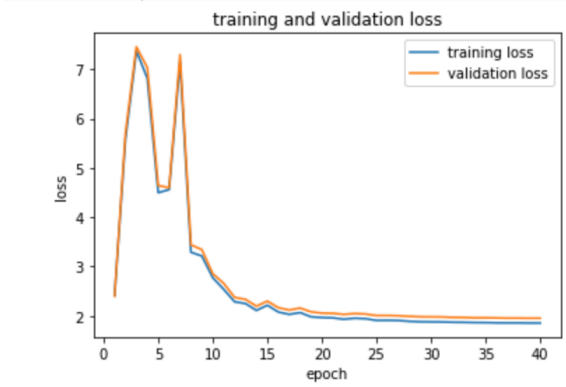
1.1 Learning rate decay

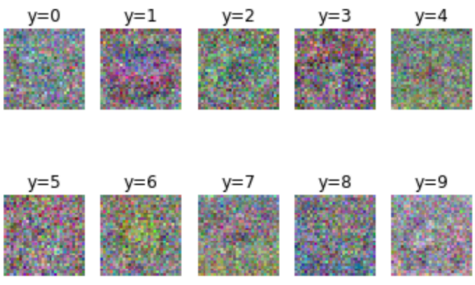
Setting a decay rate to 0.9, the learning rate is smaller after each epoch. Comparing the performance between -lambda=0, n_epochs=40, n_batch=100, eta=0.1, decay_factor=1 and -lambda=0, n_epochs=40, n_batch=100, eta=0.1, decay_factor=0.9.





the loss is 5.3867143458046 the accuracy is 0.2564





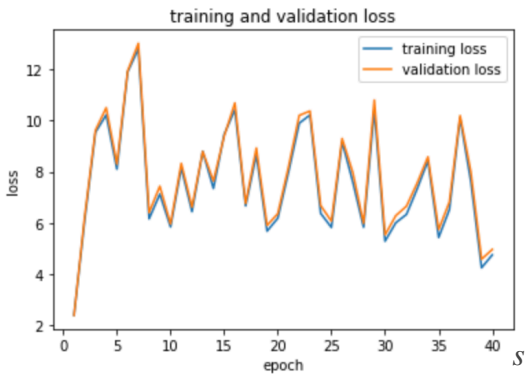
the loss is 1.9162600579362927 the accuracy is 0.3451

Decay factor	Test cost	Test accuracy
1	5.38	25.64%
0.9	1,91	34.51%

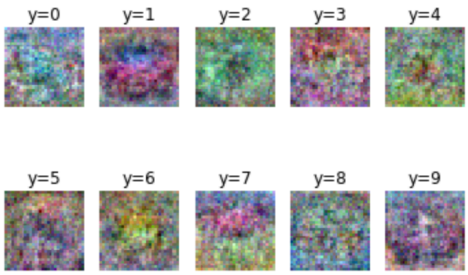
1.2 Xavier initialisation

The aim of weight initialisation is to prevent layer activation outputs from exploding or vanishing during the course of a forward pass through a deep neural network. If either occurs, loss gradients will either be too large or too small to flow backwards beneficially, and the network will take longer to converge, if it is even able to do so at all.

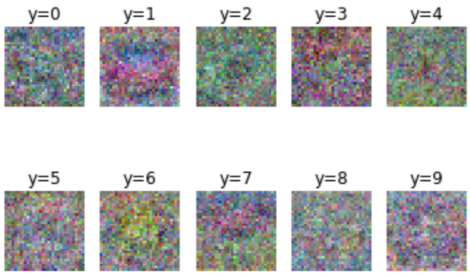
We define the std of w and b: std=sqrt(2/ni), in this place, ni denotes the input number, which is dimension of X.



$sqrt(2/ni)$




```
the loss is 5.3867143458046 the accuracy is  0.2564
```

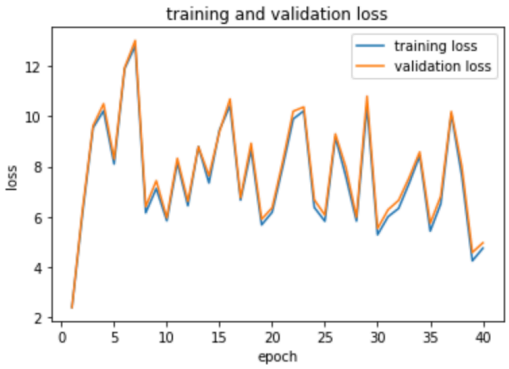



```
the loss is 4.901261948685218 the accuracy is  0.2756
```

W var	Test cost	Test accuracy
0.01	5.38	25.64%
$\sqrt{\frac{2}{ni}}$	4.90	27.56%

1.3 Shuffle training

Shuffle the order of training examples at the beginning of every epoch.




```
the loss is 5.3867143458046 the accuracy is  0.2564
```


```
the loss is 6.907854035213638 the accuracy is  0.2442
```

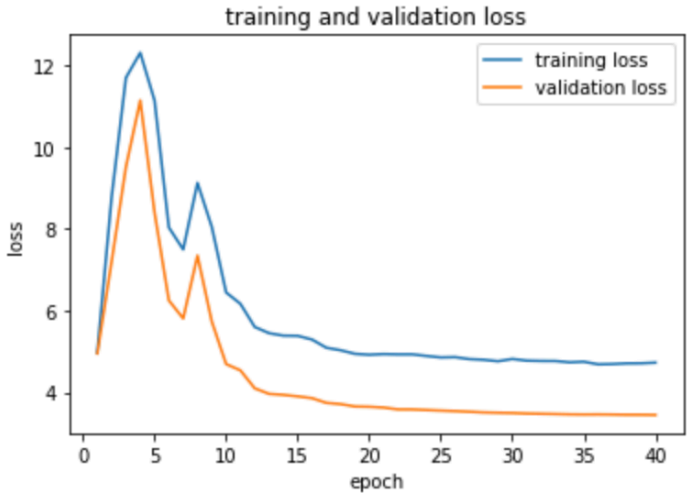
Shuffle	Test cost	Test accuracy
TRUE	5.38	25.64%
FALSE	6.90	24.42%

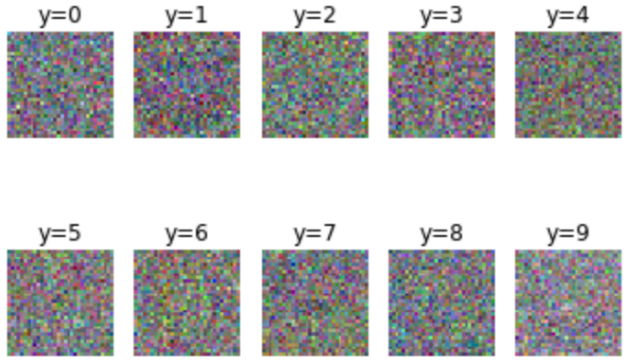
2.4 Conclusion

In my optimisation, decay factor can give the best improvement. If there is only shuffle, it seems no improvement.

The following is a network combined all the possible improvement.

-lambda=0.1, n_epochs=40, n_batch=100, eta=0.1, decay_factor=0.9 with shuffle in training data each time and Xavier initialisation.






```
the loss is 3.4271972509881037 the accuracy is  0.3164
```

2. SVM multi-class loss

I implemented sum loss function and the result is followed.

The gradient check:

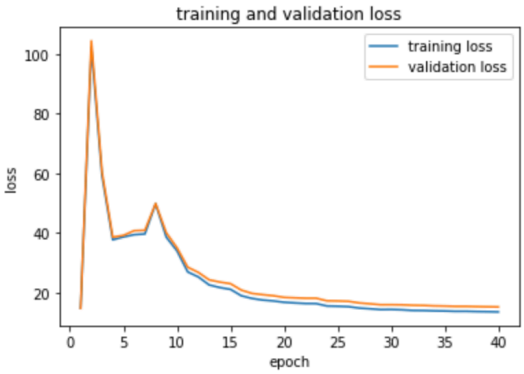
Step3:check gradients

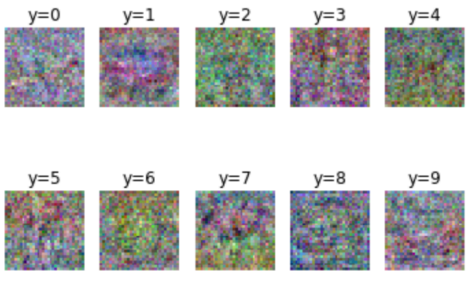
```
xmini, ymini=iterate_minibatches(Xtr.T, ytr, 2)
xmini=xmini.T
Ymini = np.eye(10, dtype=int)[ymini].transpose()
W, b= ini_parameters(xmini, Ymini)
P=evaluate_classifier(xmini, W, b)
gnw, gnb=ComputeGradsNum(X=xmini, Y= Ymini,y=ymini, P=P, W=W, b=b, h=0.0001,lamda=0)
gaw, gab=compute_gradient(X=xmini, Y= Ymini,  W=W,b=b,lamda=0,svm=True)
err=np.abs(gaw-gnw)/(np.abs(gaw)+np.abs(gnw))
print(err)
```

```
[[2.32268420e-11 9.68721194e-13 1.91181049e-11 ... 1.75322512e-12
  8.60617879e-12 1.61354724e-12]
 [7.05176492e-12 4.61340656e-12 8.08024933e-12 ... 4.60131823e-12
  1.99467626e-10 2.79873403e-11]
 [7.05176492e-12 4.61340656e-12 3.69264046e-12 ... 4.60131823e-12
  1.99467626e-10 2.79873403e-11]
 ...
 [7.05176492e-12 4.61340656e-12 8.08024933e-12 ... 1.83136132e-11
  1.99467626e-10 2.79873403e-11]
 [7.05176492e-12 9.19247101e-12 8.08024933e-12 ... 4.60131823e-12
  1.99467626e-10 5.55456604e-11]
 [1.81063899e-12 2.00597058e-11 5.14458130e-12 ... 2.49139889e-12
  2.03041484e-12 1.06600459e-12]]
```

The final accuracy:

-lambda=0.1, n_epochs=40, n_batch=100, eta=0.1, decay_factor=0.9, svm=True






```
the loss is 14.869015185902999 the accuracy is  0.3042
```