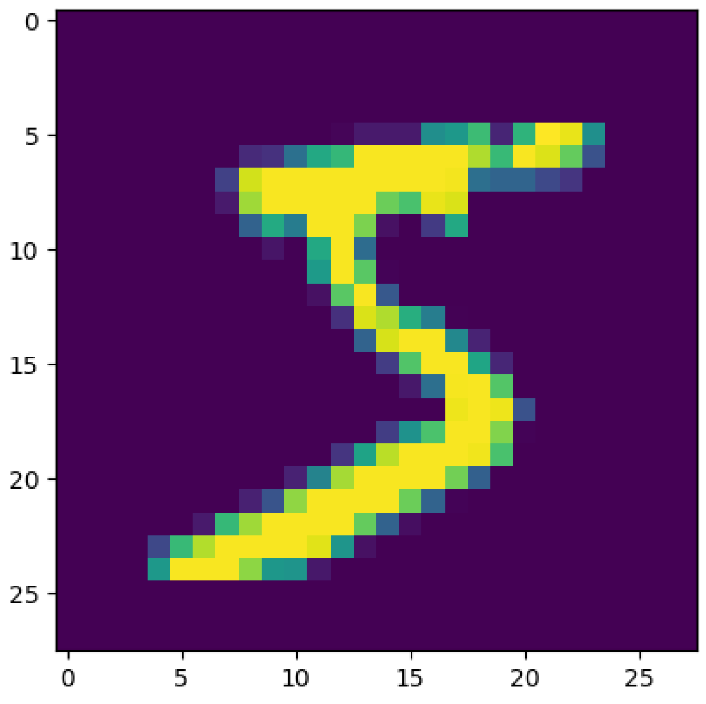
Homework1

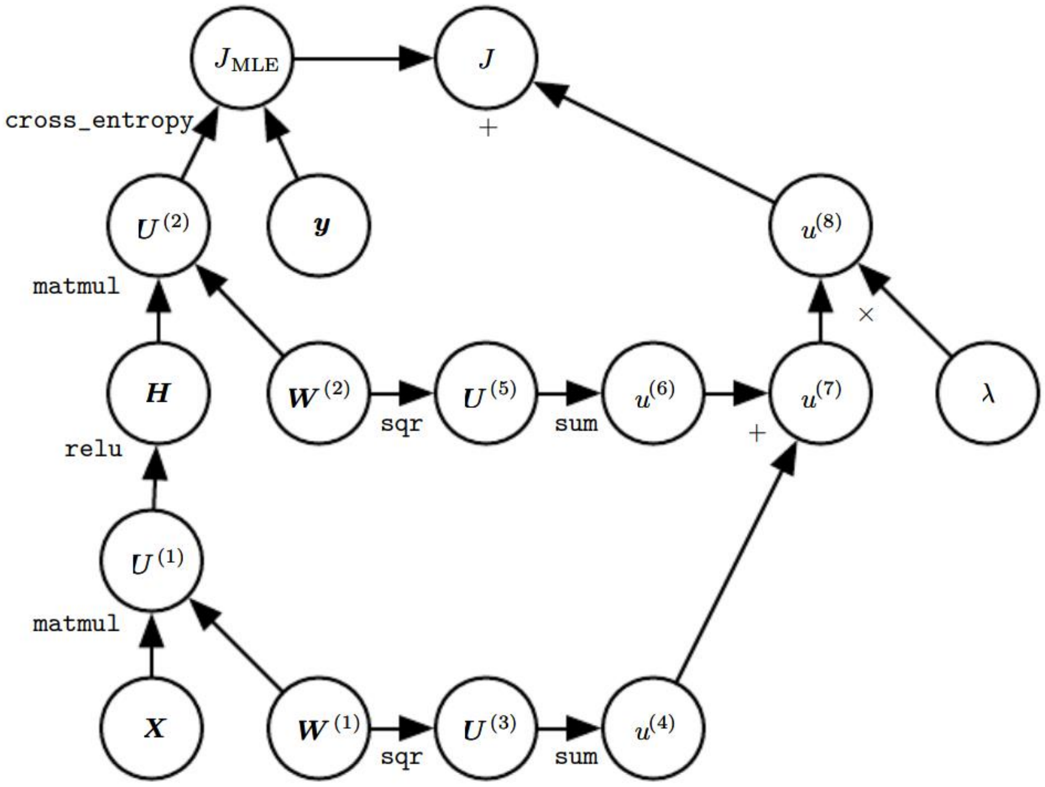
In this homework, I try to use a MLP on the Dataset MNIST. The data set contains many numbers, such as the following graph:



Training image in the MNIST

The MLP have three layer, including one hidden layer. The design of the network follows the description in the book *Deep Learning* by Ian Goodfellow, Yoshua Bengio and Aaron Courville.

The design and the back propagation are shown in the following graph. The samples, which are 1×784 vectors, are multiplied by the $W^{(1)}$, which is a 784×100 weight matrix to get $U^{(1)}$, a 1×100 vector. After relu function, the $U^{(1)}$ will become the H, with the same shape as $U^{(1)}$. H is then multiplied by $W^{(2)}$, a 100×10 weight matrix, and become $U^{(2)}$. $U^{(2)}$ is then processed by a softmax function before going into the cross entropy loss function. The loss function also contains a L2 regularization. The parameter of the regularization is 1. I also tested 0.5,5,10, however 1 is the best. The computation graph can be found in pp.220 of the *Deep Learning* book.



The computation graph of the MLP(pp.220)

$$J = J_{\mathrm{MLE}} + \lambda \left(\sum_{i,j} \left(W_{i,j}^{(1)} \right)^2 + \sum_{i,j} \left(W_{i,j}^{(2)} \right)^2 \right)$$

The loss function used in this homework

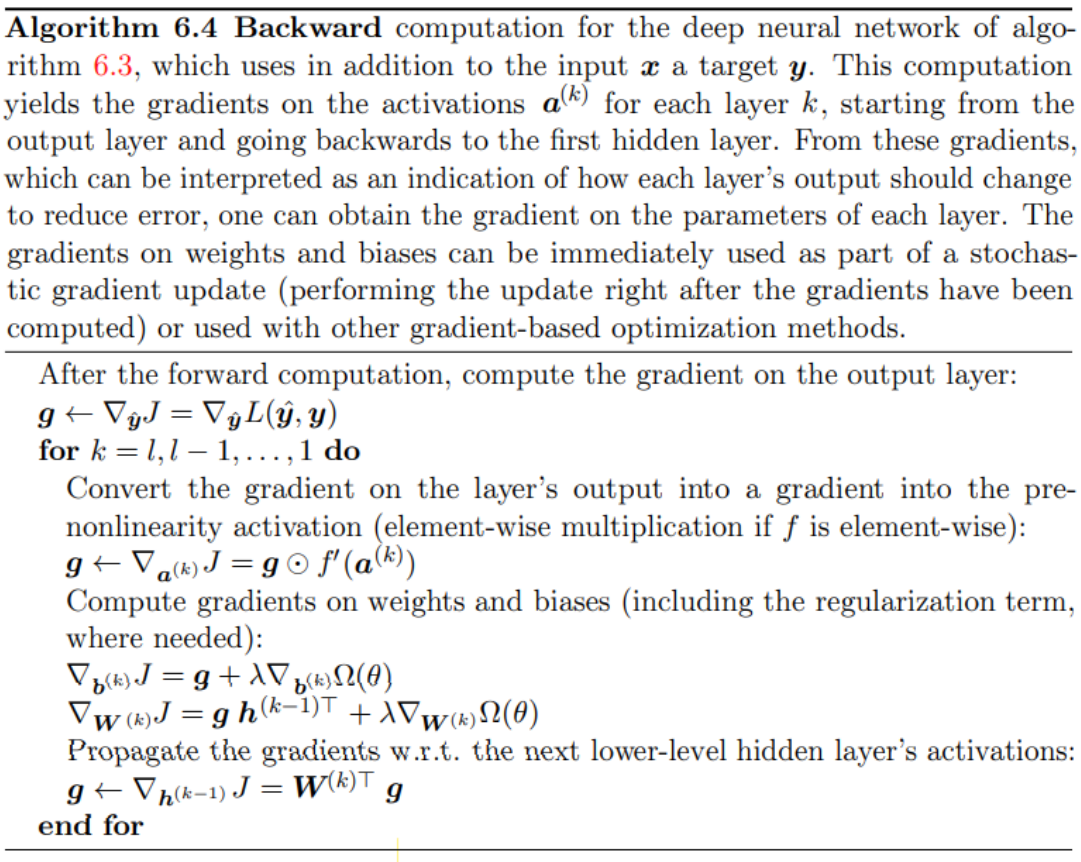
The backpropagation algorithm is as follows: For the J_{MLE} part, define the gradient with respect to $U^{(2)}$ is G. The gradient of $W^{(2)}$ is then H^TG. The backpropagation algorithm then computes $\nabla_H J$, which is $GW^{(2)}$.Next, the relu operation uses its backpropagation rule to zero out components of $GW^{(2)}$ that were less than 0. Let the result be called G' . Then the gradient of W(1) is $X^T G'$.

For the regularization part, its gradient with respect to $W^{(1)}$ is $\lambda|W^{(1)}|_2$ and for $W^{(2)}$ it is $\lambda|W^{(2)}|_2$. So the total gradient with respect to $W^{(1)}$ is:

$$X^T G' + \lambda|W^{(1)}|_2$$

While the total gradient with respect to $W^{(2)}$ is:

$$H^T G + \lambda|W^{(2)}|_2$$

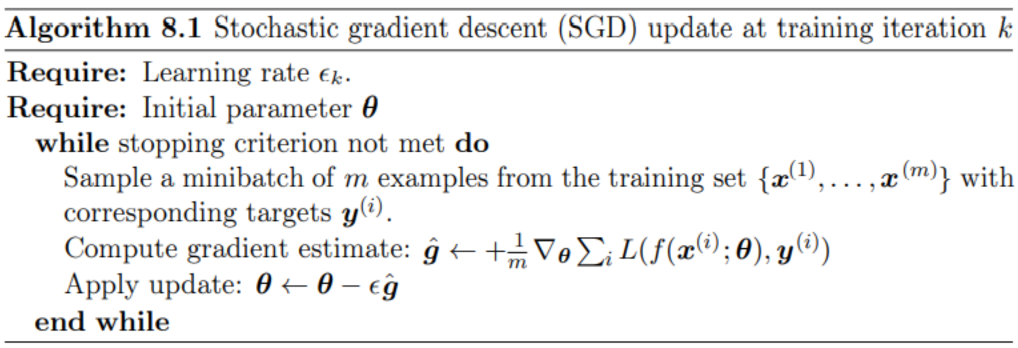


Algorithm 6.4 Backward computation for the deep neural network of algorithm 6.3, which uses in addition to the input $\boldsymbol{x}$ a target $\boldsymbol{y}$. This computation yields the gradients on the activations $\boldsymbol{a}^{(k)}$ for each layer k, starting from the output layer and going backwards to the first hidden layer. From these gradients, which can be interpreted as an indication of how each layer's output should change to reduce error, one can obtain the gradient on the parameters of each layer. The gradients on weights and biases can be immediately used as part of a stochastic gradient update (performing the update right after the gradients have been computed) or used with other gradient-based optimization methods.

```
After the forward computation, compute the gradient on the output layer:
$\boldsymbol{g} \leftarrow \nabla_{\hat{\boldsymbol{y}}} J = \nabla_{\hat{\boldsymbol{y}}} L(\hat{\boldsymbol{y}}, \boldsymbol{y})$
for $k = l, l-1, \ldots, 1$ do
    Convert the gradient on the layer's output into a gradient into the pre-nonlinearity activation (element-wise multiplication if $f$ is element-wise):
    $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{a}^{(k)}} J = \boldsymbol{g} \odot f'(\boldsymbol{a}^{(k)})$
    Compute gradients on weights and biases (including the regularization term, where needed):
    $\nabla_{\boldsymbol{b}^{(k)}} J = \boldsymbol{g} + \lambda \nabla_{\boldsymbol{b}^{(k)}} \Omega(\theta)$
    $\nabla_{\boldsymbol{W}^{(k)}} J = \boldsymbol{g}\, \boldsymbol{h}^{(k-1)\top} + \lambda \nabla_{\boldsymbol{W}^{(k)}} \Omega(\theta)$
    Propagate the gradients w.r.t. the next lower-level hidden layer's activations:
    $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{h}^{(k-1)}} J = \boldsymbol{W}^{(k)\top} \boldsymbol{g}$
end for
```

Backpropagation algorithm used in the homework(pp.213)

The SGD algorithm is as follows:



Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

```
Require: Learning rate $\epsilon_k$.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
  end while
```

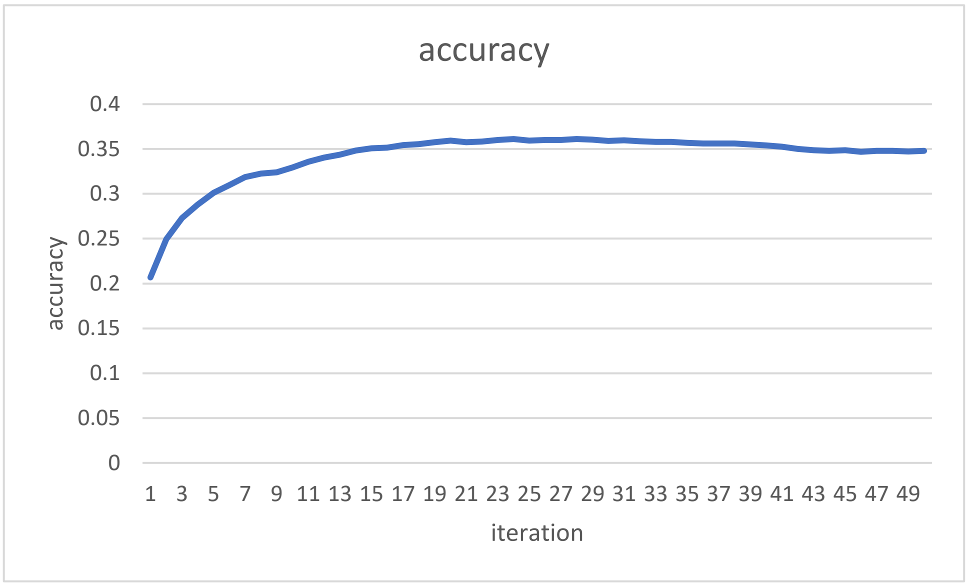
SGD algorithm, in pp.294

I use 50 as the max iteration, norm of the gradient of $W^{(1)}$ and $W^{(2)}$ both less than 0.1 as the stopping criterion, and 100 as the batch size.

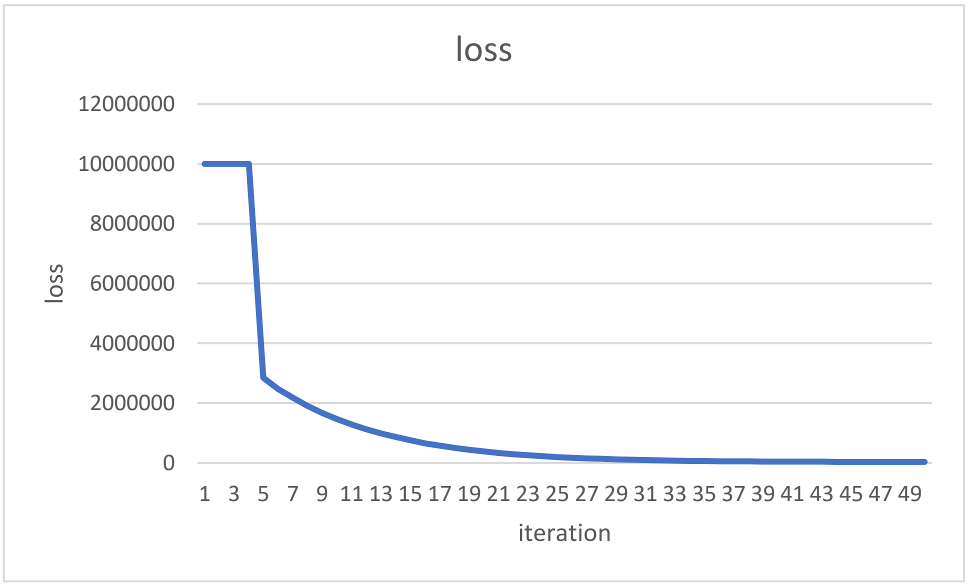
For learning rate, I follows the following equation in pp.295:

$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau$$

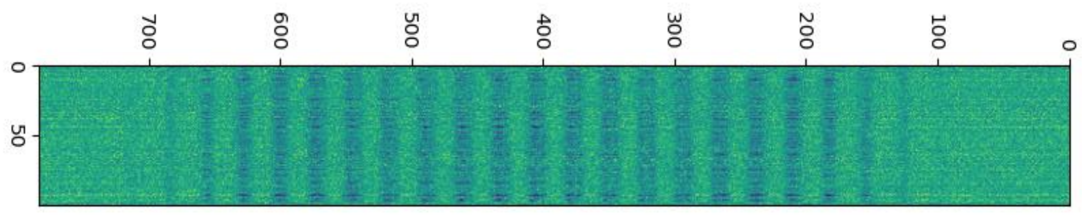
The ϵ_k is the learning rate for k iteration,τ is the max number of iterations, α is $\frac{k}{\tau}$,ϵ_0 is the initial learning rate, and ϵ_τ is the final learning rate. I choose 0.001 (I also tested 0.01 and 0.1, 0.001 works best) as the initial learning rate, and 0.0005 (0.001 and 0.01 are also tested) as the final learning rate.

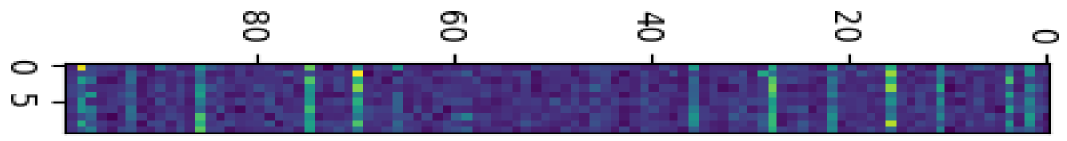


In the training process, after every iteration, I record the accuracy of the network, which is defined as the number of the truly predicted samples in the test set divided by the number of all samples in the test set. The network reaches the accuracy of approximate 0.36 after 23 iterations, and then become less accurate, at last, the accuracy of the network after 50 iterations is 0.34.



The loss after the first three iterations are NaN, the computer encounter overflow problems. It then reduce quickly until iteration 23.





The two graphs above are the illustration of weight1 and weight2.