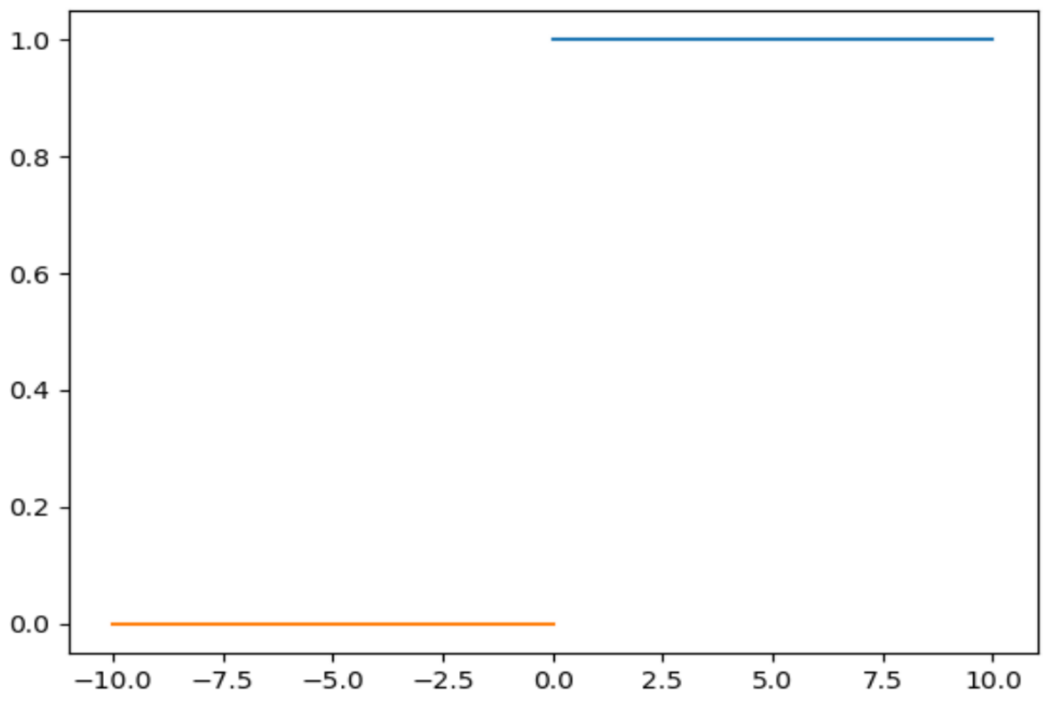
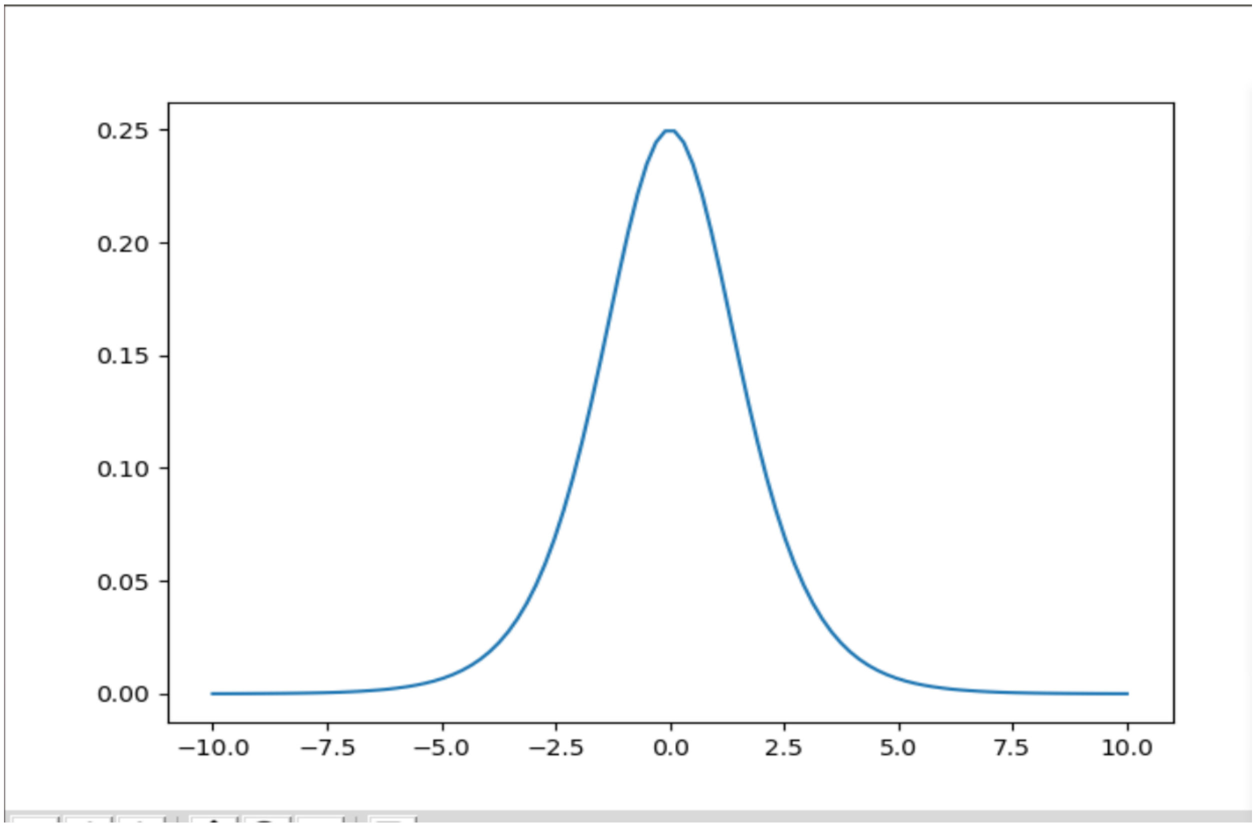
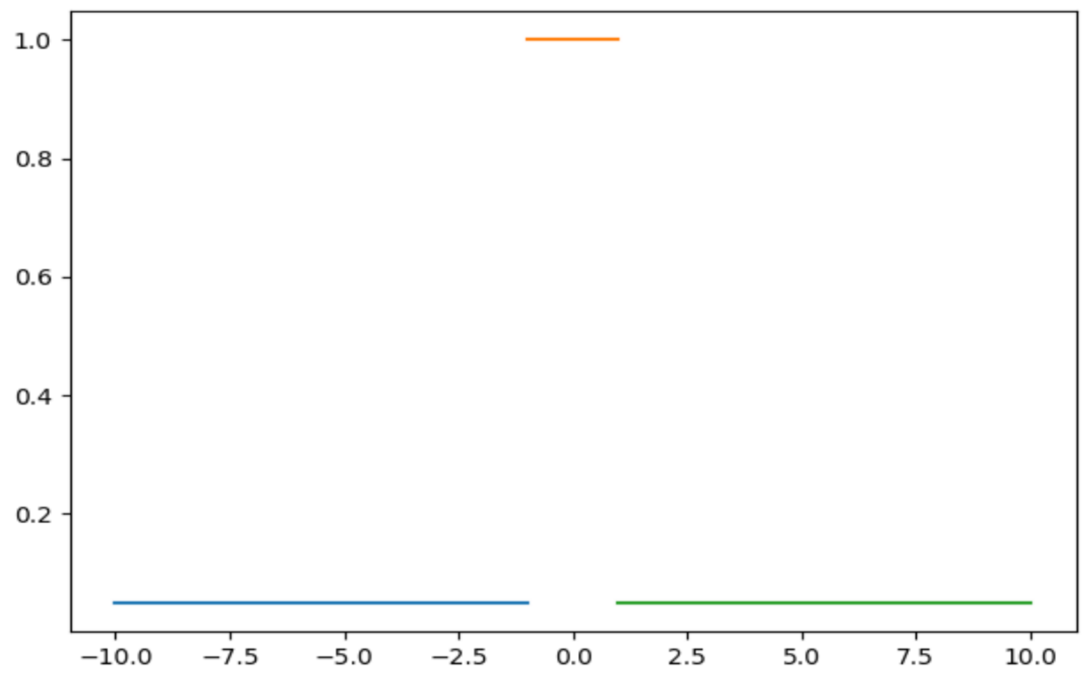
Problem 1:



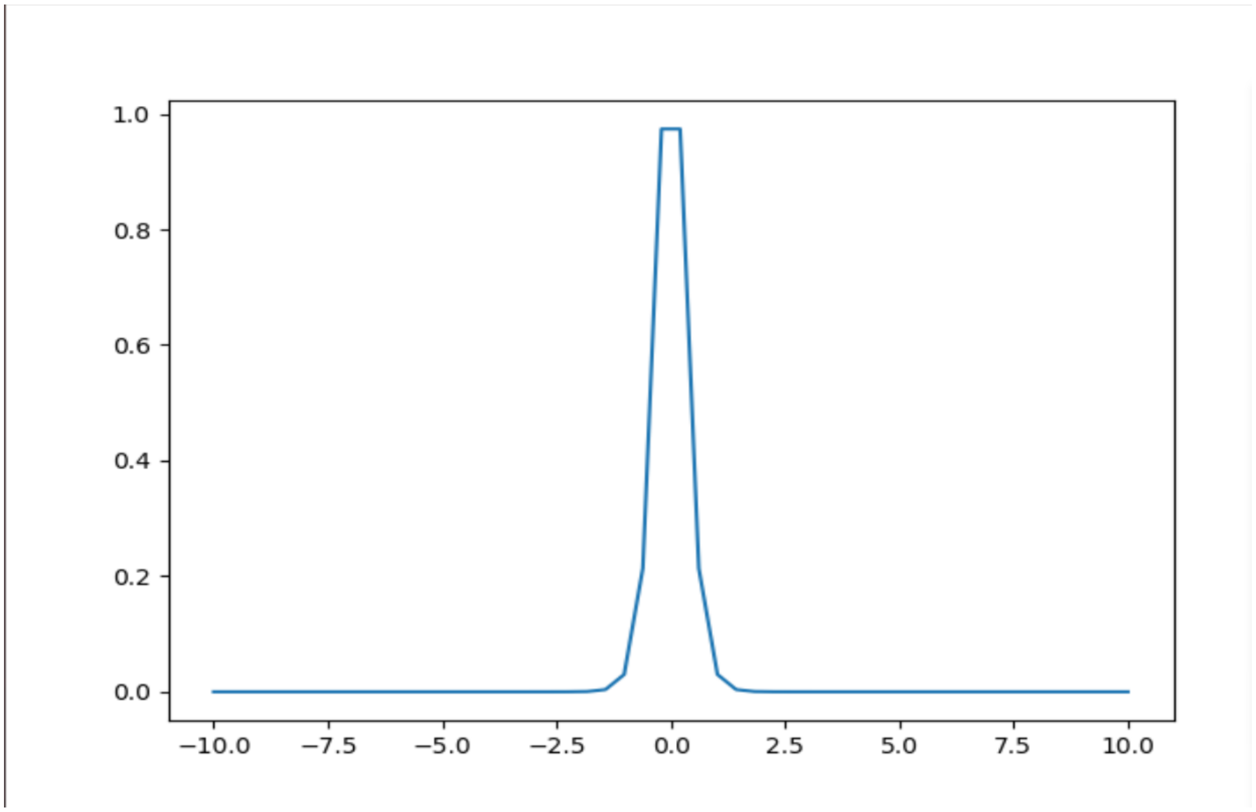
This is Gradient of Rectified linear unit function. When x is range [-10,0), it’s inactive learning region. (yellow line) when x is range [0,10], it’s fast learning region. (blue lines)



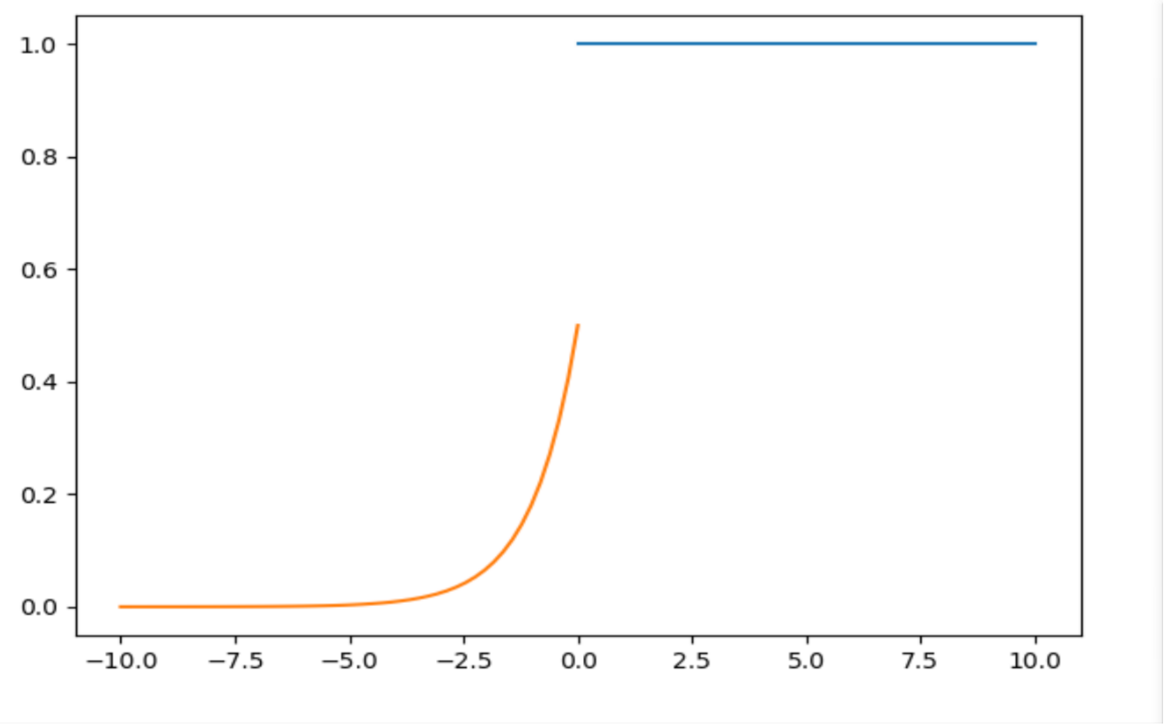
This is gradient of Logistic sigmoid activation function. When x is range [-10,-4.54), and (4.54,10] it's slow learning. When x is range [-4.54,4.54], it's active learning.



This is the gradient of Piece-wise linear unit function. When x is range [-1,1], it’s fast learning. When x is range [-10, -1] and [1,10], it’s active learning.



It’s the gradient of Swish function. When x is range [-10, -1.02) (1.02,10], it’s slow learning. When x is range (-1.02,1.02), it’s active learning. When x is (-0.2, 0.2), it’s fast learning.



It’s the gradient of exponential linear unit function. When x is range [-10, -3.87], it’s slow learning. When x is range (3.87,0), it’s active learning. When x is [0,10], it’s fast learning.

Problem 2:

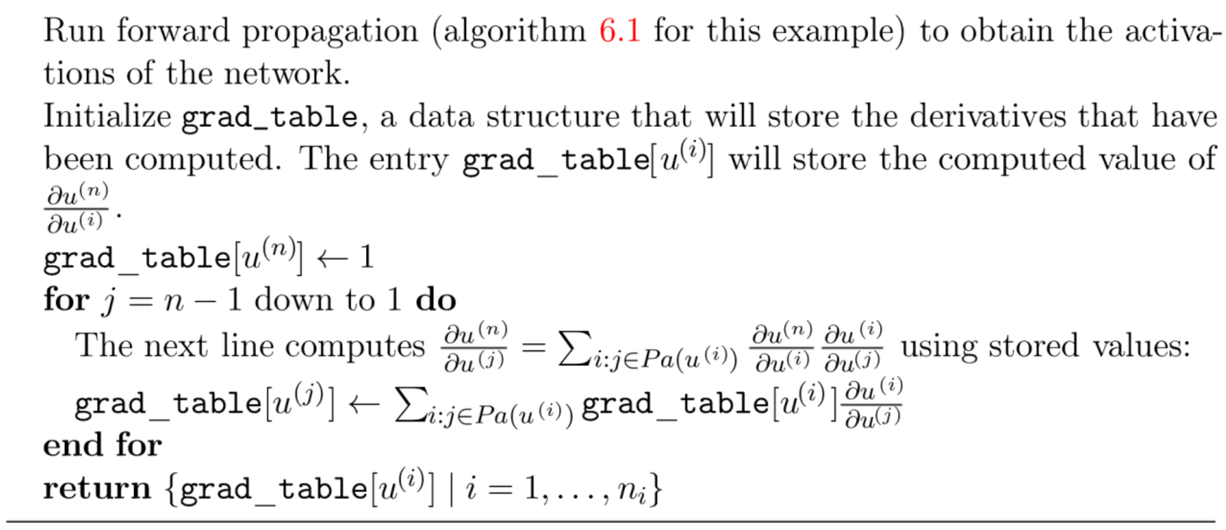
```
for $i = 1, \ldots, n_i$ do
    $u^{(i)} \leftarrow x_i$
end for
for $i = n_i + 1, \ldots, n$ do
    $\mathbb{A}^{(i)} \leftarrow \{u^{(j)} \mid j \in Pa(u^{(i)})\}$
    $u^{(i)} \leftarrow f^{(i)}(\mathbb{A}^{(i)})$
end for
return $u^{(n)}$
```

The algorithm 6.1 is the forward propagation.

The line 1 to line 3: means mapping input nodes u (1) u(ni) to an output u(n), which describe how to compute a single scalar u(n) because we want to obtain the gradient of this scalar.

The line 4 to line6: u(j) means calculating and weighting i-th node. A(i) means the calculated value is given to A(i). u(i) means by active function, recalculate u(i) value. The for loop means: one by one, after calculating the i-th layer, calculating i+1-th layer. Each node u(i) is associated with an operation f(i) and it's computed by evaluation function f(A(i)), which is the set of all nodes that are parents of u(i).

The final step: return u(n): after calculating every layer, returning final value, that is output because we need to get final gradient.



```
Run forward propagation (algorithm 6.1 for this example) to obtain the activations of the network.
Initialize grad_table, a data structure that will store the derivatives that have been computed. The entry grad_table[$u^{(i)}$] will store the computed value of $\frac{\partial u^{(n)}}{\partial u^{(i)}}$.
grad_table[$u^{(n)}$] $\leftarrow 1$
for $j = n - 1$ down to 1 do
  The next line computes $\frac{\partial u^{(n)}}{\partial u^{(j)}} = \sum_{i:j \in Pa(u^{(i)})} \frac{\partial u^{(n)}}{\partial u^{(i)}} \frac{\partial u^{(i)}}{\partial u^{(j)}}$ using stored values:
  grad_table[$u^{(j)}$] $\leftarrow \sum_{i:j \in Pa(u^{(i)})}$ grad_table[$u^{(i)}$] $\frac{\partial u^{(i)}}{\partial u^{(j)}}$
end for
return {grad_table[$u^{(i)}$] $\mid i = 1, \ldots, n_i$}
```

First defines the grad_table: it's to store derivative value.

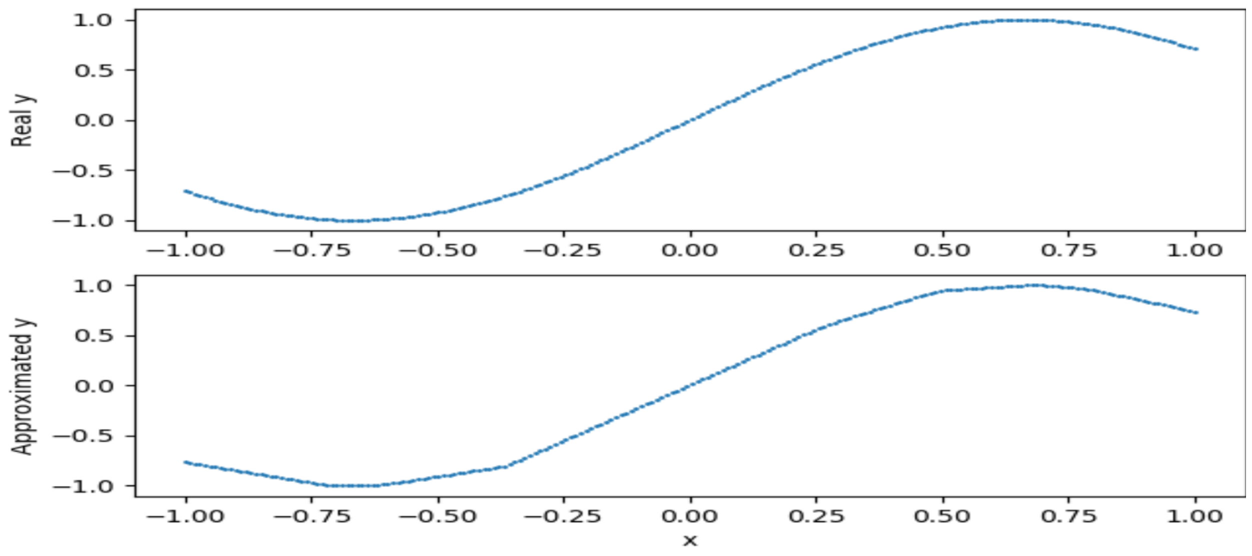
The next we need to calculate new derivative (u(n)/ u(j)) one by one because it can avoid the exponential explosion in repeated subexpressions. Every node of the compute derivative u(n)/u(i) associated with the forward node u(i), which can be defined next. And a dot product can be used for each node between the gradient already computed with respect to nodes u(i) that are children of u(j). moreover, this vector containing the partial derivative for some u(i).

Finally, we output the new gradient table, so as finish the backpropagation.

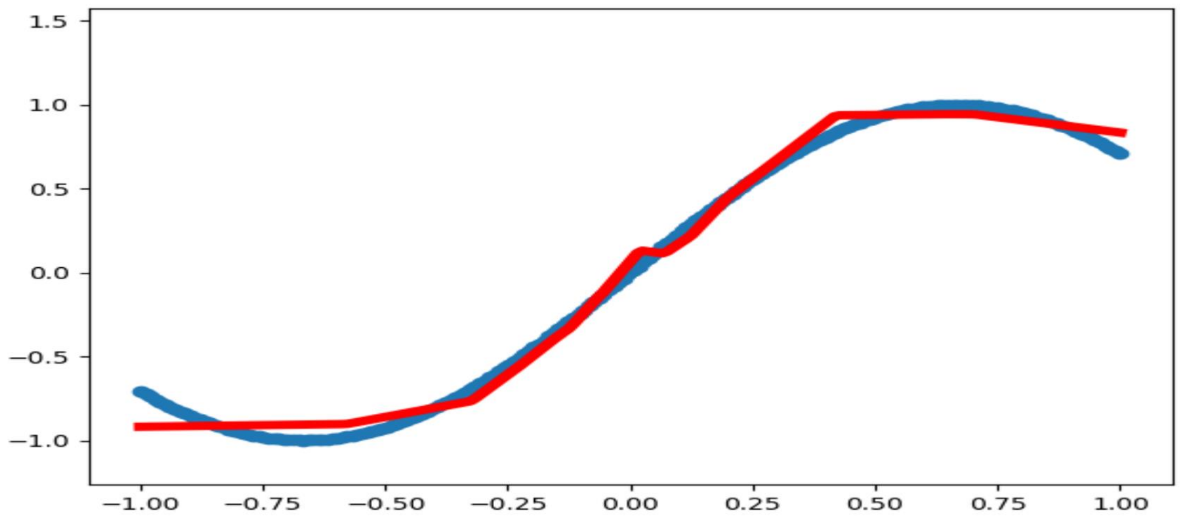
(2)

For some cases, it is the wasteful to computing the same subexpression twice. Especially in complicated algorithm, there can be exponentially many of these wasted computations, which make the chain rule infeasible. And for other cases, it is a good way to reduce memory consumption at the cost of higher runtime, if calculating the same subexpression.

Problem 3:



I use Keras to construct neutron network. And train for 7000 times. So get the similar curve.



And also I use tensorflow to test. When hidden neutron is 10, it gets the function slower. But when setting the neutron is 20, it will be faster. And I train for 1000 times.

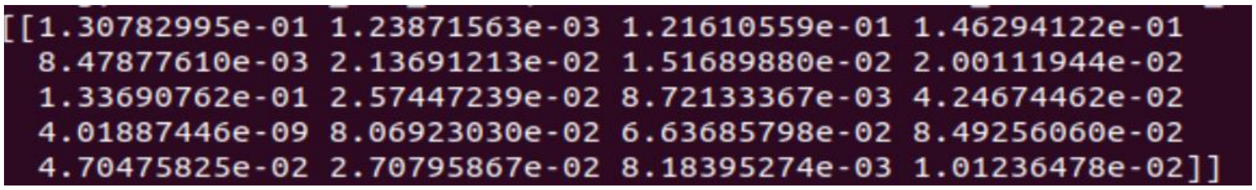
Problem 4:

(1)

The architecture of the softmax layer: input feature:100 (asample is [100,20], 20 neutons (bias is [1,20]).

We get 100 input features to 20 neurons, so the total number of connections is 100x20 =2000

(2)

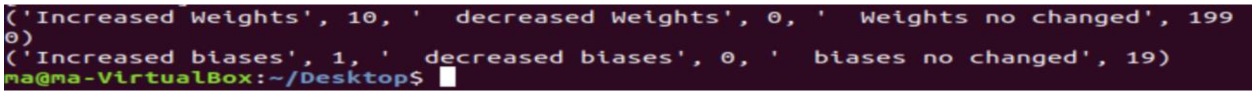


```
[[1.30782995e-01 1.23871563e-03 1.21610559e-01 1.46294122e-01
  8.47877610e-03 2.13691213e-02 1.51689880e-02 2.00111944e-02
  1.33690762e-01 2.57447239e-02 8.72133367e-03 4.24674462e-02
  4.01887446e-09 8.06923030e-02 6.63685798e-02 8.49256060e-02
  4.70475825e-02 2.70795867e-02 8.18395274e-03 1.01236478e-02]]
```

According to the result, 0.14 is the biggest probability. So, it's 4th class.

(3)

This is the number of the weights and biases will be increased, decreased, or remain the same respectively.



```
('Increased Weights', 10, '  decreased Weights', 0, '  Weights no changed', 199
0)
('Increased biases', 1, '  decreased biases', 0, '  biases no changed', 19)
ma@ma-VirtualBox:~/Desktop$ 
```

I tried a lot of time to get the dl/dw and dl/db. I don't know if it's correct.

I got the number of increased weights is 10, the number of decreased weights is 0, so there is 1990 Weights that have no changes.

And only one bias is increased and no one bias is decreased, so there is 19 biases that have no changes.

(4). Not finish

Problem 5

$$\log p(w|w_I) = \log \sigma({v'_w}^T v_{w_I}) + \sum_{i=k}^{K} E_{w_i \sim P_n(w)}[\log \sigma(-{v'_w}^T v_{w_I})]$$

(1)

This is the objective function for one sample. Instead of changing all the weights each time, we just use k of them and increase computational efficiency dramatically. And it's depends on the number of negative samples. The difference from stochastic gradient descent is the fact that we're taking not only one observation into account but k of them.

And we use noise distribution to get the probability distribution.

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n}(f(w_j)^{3/4})}$$

This is the noisy formula where ¾ is the value found by taking experiments; f(w) is the frequency of the word in the corpus.

(2)

$$\begin{aligned}\frac{\partial E}{\partial \mathbf{h}} &= \sum_{w_j \in \{w_O\} \cup \mathcal{W}_{\text{neg}}} \frac{\partial E}{\partial \mathbf{v}'_{w_j}{}^T \mathbf{h}} \cdot \frac{\partial \mathbf{v}'_{w_j}{}^T \mathbf{h}}{\partial \mathbf{h}} \\ &= \sum_{w_j \in \{w_O\} \cup \mathcal{W}_{\text{neg}}} \left(\sigma(\mathbf{v}'_{w_j}{}^T \mathbf{h}) - t_j\right) \mathbf{v}'_{w_j} := \text{EH}\end{aligned}$$

(3)