Deep Learning

Final exam (take home) : May 28–Due June 11

Name : 정 상 만 Student Number : 20190310290

Problem 1.

For the XOR function, we want to find the following form of a linear model

$$f(x;w,b)=x^Tw+b$$

that minimizes the norm,

$$J(\theta)=\frac{1}{4}\sum_{x\in X}\left(f^*(x)-f(x;\theta)\right)^2.$$

For the above problem, show that the optimal parameter is

$$w=0 \text{ and } b=\frac{1}{2}.$$

For the XOR function, we can propose to find the following form of a nonlinear model

$$f(x;W,c,w,b)=w^T\max\{0,\ W^Tx+c\}+b.$$

Show that the optimal parameter is

$$W=\begin{bmatrix}1&1\\1&1\end{bmatrix},\ c=\begin{bmatrix}0\\-1\end{bmatrix},\ w=\begin{bmatrix}1\\-2\end{bmatrix},\text{ and } b=0.$$

Solution)

Note that $\vec{x}=[x_1\ x_2]^T\in X=\{[0\ 0]^T,[0\ 1]^T,[1\ 0]^T,[1\ 1]^T\}$ and XOR function is defined by $f^*=0$ if $x_1=x_2$, and $f^*=1$ if $x_1\neq x_2$. We want to find the parameter θ that minimizes the loss function $J(\theta)$. This can be calculated as formulating **the normal equation**. Thus, we obtain the results as :

Let $\nabla_{\vec{w}}J(\theta)=\begin{bmatrix}\frac{\partial J}{\partial w_1}\\ \frac{\partial J}{\partial w_2}\end{bmatrix}=\vec{0}$ and $\frac{\partial J}{\partial b}=0$, then $\frac{\partial J}{\partial w_1}=w_1+\frac{1}{2}w_2+b=\frac{1}{2}$, $\frac{\partial J}{\partial w_2}=\frac{1}{2}w_1+w_2+b=\frac{1}{2}$ and $\frac{\partial J}{\partial b}=2b+w_1+w_2=1$. These can be solved $w_1=w_2=0\ \Rightarrow\ \vec{w}=\vec{0}$ and $b=\frac{1}{2}$.

For the nonlinear function $f(x;W,c,w,b)=w^T\max\{0,\ W^Tx+c\}+b$, put the given optimal parameters of f, then the cost function $J(\theta)=0$. Thus the parameters are optimal.

(*** In this case, we have to update the parameters using **the gradient descent method** to obtain the minimum for the cost function and using **back-propagation** to keep going to learn iteratively until the parameters are optimal. Refer to **problem 4**. ***)

Refer to the following codes using MATLAB.

```
clear, clc
% data
X=[0 0; 0 1; 1 0; 1 1]';
% true XOR function values
true_f=zeros(1,4);
for i=1:4
    true_f(i)=XOR(X(:,i));
end
% our model
model_f=@(x,W,c,w,b) w'*max(0,W'*x+c)+b;
% random & optimal parameters
iW=rand(2,2); ic=rand(2,1); iw=rand(2,1); ib=rand;
W=ones(2,2); c=[0 -1]'; w=[1 -2]'; b=0;
% compute the cost function
cost_J_random=1/4*sum((true_f-model_f(X,iW,ic,iw,ib)).^2)
cost_J_optimal=1/4*sum((true_f-model_f(X,W,c,w,b)).^2)

>>
cost_J_random =

    0.2958


cost_J_optimal =

     0
```

■

Problem 2.

In feedforward network models, various activation functions can be used. Discuss about the pros and cons of the following activation functions: Relu, logistic sigmoid, soft plus, and hyperbolic tangent.

Solution)

The functions **Relu, sigmoid, soft plus** and **hyperbolic tangent** are defined as follows.

$f(x)=\max\{0,x\}$, $f(x)=\frac{1}{1+e^{-x}}$, $f(x)=\ln(1+e^{x})$ and $f(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$, respectively.

This activation functions have the desirable properties such as :

1. Non-linearity, 2. Appropriate Range, 3. Continuously differentiable, 4. Monotonic, 5. Smooth with a monotonic derivative, 6. Approximates identity near the origin.

These because :

1. We want to design the layer more depth in order to get the nicely expected data obtained by a function as we now approximate it. However, if the activation function is linear, and assume that the neural network has the three hidden layer. Then we encounter the problem :

Define $f^{(1)}(x)=ax+b$, $f^{(2)}(s)=cs+d$, $f^{(3)}(t)=et+g$, and NN is $y=f^{(3)}(f^{(2)}(f^{(1)}(x)))$, then we can write $y=e(c(ax+b)+d)+g=(ace)x+(bc+ed+g)$. Let $ace=\alpha$ and $bc+ed+g=\beta$, then this model can be written as $y=\alpha x+\beta$ and hence this model is reduced the neural network formed one layer.

2. In order for updating the weights and bias automatically, we have to consider the backpropagation. This is the method that finds the optimal value of the parameters (weights, bias and so on in neural network) for each trial to update and reflect the error between the data and model using gradient descent method or others. In specific, the gradient descent method uses differentiation. When the range of the activation function is finite, gradient-based training methods tend to be more stable, because pattern presentations significantly affect only limited weights. When the range is infinite, training is generally more efficient because pattern presentations significantly affect most of the weights. In the latter case, smaller learning rates are typically necessary.

3. This property is desirable for enabling gradient-based optimization methods. The binary step activation function is not differentiable at 0, and it differentiates to 0 for all other values, so gradient-based methods can make no progress with it.

4. When the activation function is monotonic, the error surface associated with a single-layer model is guaranteed to be convex.

5. These have been shown to generalize better in some cases.

6. When activation functions have this property, the neural network will learn efficiently when its weights are initialized with small random values. When the activation function does not approximate identity near the origin, special care must be used when initializing the weights, i.e. activation function where $f(0)=0$ and $f'(0)=1$ and f' is countinuous at 0 are indicated as having this property.

Now, let me see the pros and cons for the activation functions above.

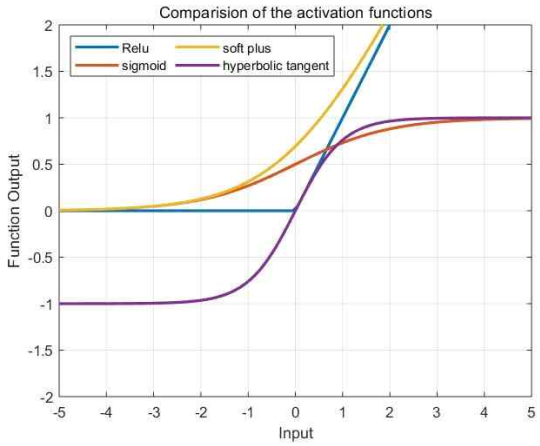


Figure 1. Comparison of the activation functions.

In this figure, all of them are non-linear. Thus we can ready to use those functions. But there are many differences between these functions.

[Relu] ⇒ **pros** : 1, 2, 4 and 5, / **cons** : 3 and 6.
[Sigmoid] ⇒ **pros** : 1, 3, 4, and smooth. / **cons** : 2, 5 (monotonic derivative), 6.
[Soft plus] ⇒ **pros** : 1, 2, 3, 4 and 5, / **cons** : 6.
[Hyperbolic tangent] ⇒ **pros** : 1, 3, 4 / **cons** : 2, 5, 6.

The above figure obtained by using this MATLAB code.

```
% Activation functions
clear,clc
x=linspace(-5,5,100);
sigmoid=@(x) 1./(1+exp(-x));
softplus=@(x) log(1+exp(x));
tangentH=@(x) (exp(2*x)-1)./(exp(2*x)+1);
plot(x,Relu(x),x,sigmoid(x),x,softplus(x),x,tangentH(x),'LineWidth',2); grid
axis([-5 5 -2 2]);
legend({'Relu','sigmoid','soft plus','hyperbolic tangent'},'Location',
'northwest','NumColumns',2)
xlabel('Input');ylabel('Function Output');
title('Comparision of the activation functions')
```

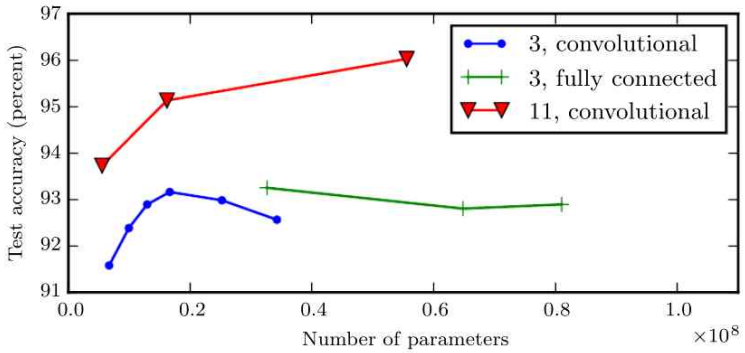
Relu.m

```
function r=Relu(x)
r=max(0,x);
```

■

Problem 3.

The following figure, Figure 6.7 in the book, gives some insight how to design feedforward network. Discuss the design issue related to the depth and the number of parameters.



Solution)

If the depth (the number of the layer) is the same, two networks seem to be a little bit differ for the accuracy. That is, the number of parameters doesn't affect the accuracy but the depth do. See the curve of the convolutional network that has 11 layers. Under the same networks, 11 depth case is pretty better than 3 depth case. This graph tells us the accuracy is determined by the depth rather than the number of parameters.

■

Problem 4.

Write down a computational graph for the following function :

$$f(x\,;W,c,w.b) = w^T \max\{0,\ W^T x + c\} + b$$

Write down an algorithm how to compute the gradient of $f(x;\theta)$ using a back-propagation (or you can draw a computational graph for this).

Solution)

Using back-propagation and computational graph, we can compute the gradient of $f(\vec{x};\theta)$ as :

$$\nabla_W f(\vec{x};\theta) = \frac{\partial f}{\partial w_{ji}},\ \nabla_{\vec{w}} f(\vec{x};\theta) = \frac{\partial f}{\partial w_i},\ \nabla_{\vec{c}} f(\vec{x};\theta) = \frac{\partial f}{\partial c_i},\ \frac{\partial f}{\partial b},\ i=1,2 \text{ and } j=1,2.$$

Let $u^{(1)} = W^T\vec{x}$, $u^{(2)} = u^{(1)} + \vec{c}$, $u^{(3)} = \max\{0, u^{(2)}\}$, $u^{(4)} = \vec{w}^T u^{(3)}$, $f = u^{(4)} + b$. Then the gradients are computed by using chain rule as follows :

$$\nabla_W f(\vec{x};\theta) = \frac{\partial f}{\partial w_{ji}} = \frac{\partial f}{\partial u_i^{(4)}}\frac{\partial u_i^{(4)}}{\partial u_i^{(3)}}\frac{\partial u_i^{(3)}}{\partial u_i^{(2)}}\frac{\partial u_i^{(2)}}{\partial u_i^{(1)}}\frac{\partial u_i^{(1)}}{\partial w_{ji}} = (1)(w_i)(\max\{0, u_i^{(2)}\})(1)(x_j),$$

$$\nabla_{\vec{w}} f(\vec{x};\theta) = \frac{\partial f}{\partial w_i} = \frac{\partial f}{\partial u_i^{(4)}}\frac{\partial u_i^{(4)}}{\partial w_i} = (1)u_i^{(3)},$$

$$\nabla_{\vec{c}} f(\vec{x};\theta) = \frac{\partial f}{\partial c_i} = \frac{\partial f}{\partial u_i^{(4)}}\frac{\partial u_i^{(4)}}{\partial u_i^{(3)}}\frac{\partial u_i^{(3)}}{\partial u_i^{(2)}}\frac{\partial u_i^{(2)}}{\partial c_i} = (1)(w_i)(\max\{0, u_i^{(2)}\})(1),$$

$$\frac{\partial f}{\partial b} = 1.$$

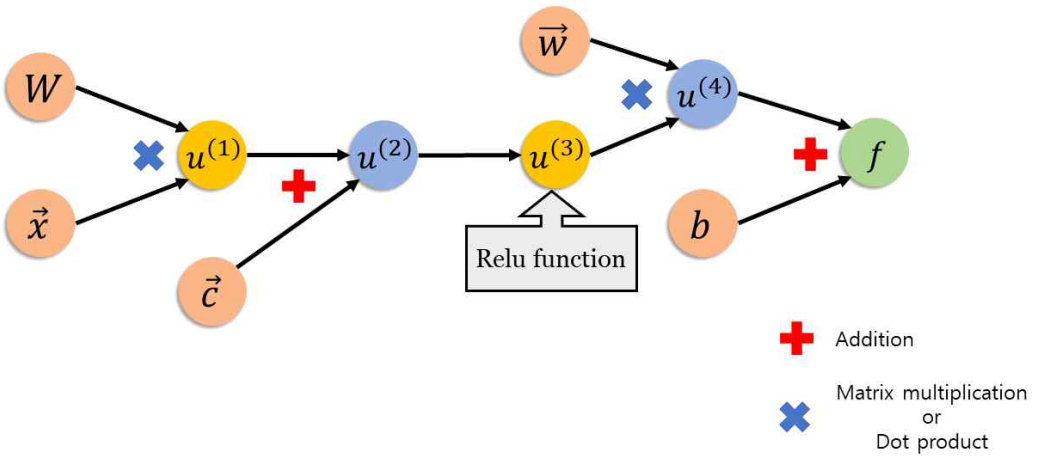


Figure 2. Computational graph for forward propagation

■

Problem 5.

Consider the following 3^{rd} order polynomial,

$$y = 1 + x + x^2 + x^3$$

and generate 100 data points (x_i, y_i) randomly, with $y_i = f(x_i)$, $i = 1, \cdots, 100$ and $x_i \in (0, 5)$.

Form a feedforward network of the following form,

$$f(x;\theta) = W_3\,\sigma(W_2\,\sigma(W_1 x + b_1) + b_2) + b_3$$

where $\sigma(x)$ is an activation function given by $\sigma(x) = \dfrac{1}{1+e^{-x}}$.

We want to find the optimal parameters $\theta = W_1, W_2, W_3, b_1, b_2, b_3$ for the following cost function:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(y_i - f(x_i;\theta)\right)^2 + \lambda\left(W_1^2 + W_2^2 + W_3^2\right)$$

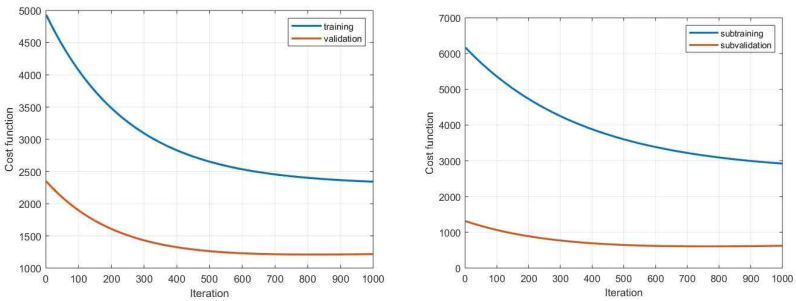
where m is the number of data in the training set and $\lambda = 10$.

Step 1: Divide the data set into two groups, training data set (60 data points) and validation data set (40 data points). Use gradient descent with zero initial parameters, learning rate $\epsilon = 10^{-5}$ and use back-propagation algorithm for the calculation of the gradient of $J(\theta)$.
Update the parameters for 1000 iterations, plot values of $J(\theta)$ at each iteration. At the same figure, plot the following error for the validation data set, $E(\theta) = \dfrac{1}{m}\sum_{i=1}^{m}\left(y_i - f(x_i;\theta)\right)^2$, where m is the number of data in the validation set. (If the number of iteration is not enough for the training, you can iterate more. You can also try to use a better learning rate.)

Step 2: Divide the training data set (60 data points) into two groups, subtrain data (40data) and validation data (20data), run the early stopping algorithm in Algorithm 7.1 and find the number of training, i^*, and perform the Step 1 up to i^* iterations.

Discuss about the results obtained in Step 1 and Step 2. When λ is much smaller or much bigger than 10, $\lambda = 10^{-3}$ or $\lambda = 10^4$, discuss about the result you will obtain from Step 2.

Solution)



■