ECE 449 Machine Learning (21Fa): Assignment 3

Xu Ke 3190110360

Nov. 20, 2021

1 Problem 1: CNN

The problem already gives us that the expression of $\frac{\partial loss}{\partial P}$, which is:

$$\frac{\partial loss}{\partial P} = \begin{bmatrix} g_{1,1} & g_{1,2} & \cdots & g_{1,m} \\ g_{2,1} & g_{2,2} & \cdots & g_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ g_{n,1} & g_{n,2} & \cdots & g_{n,n} \end{bmatrix} \tag{1}$$

For the 2×2 AvgPooling Layer, in the back-propogation process, an element in one grid will be divided into 6 peices, thus the gradient $\frac{\partial loss}{\partial S}$ is:

$$\frac{\partial loss}{\partial S} = \begin{bmatrix} \frac{g_{1,1}}{4} & \frac{g_{1,1}}{4} & \cdots & \frac{g_{1,m}}{4} & \frac{g_{1,m}}{4} \\ \frac{g_{1,1}}{4} & \frac{g_{1,1}}{4} & \cdots & \frac{g_{1,m}}{4} & \frac{g_{1,m}}{4} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{g_{n,1}}{4} & \frac{g_{n,1}}{4} & \cdots & \frac{g_{n,m}}{4} & \frac{g_{n,m}}{4} \\ \frac{g_{n,1}}{4} & \frac{g_{n,1}}{4} & \cdots & \frac{g_{n,m}}{4} & \frac{g_{n,m}}{4} \end{bmatrix} \tag{2}$$

which implies the expression using $g_{i,j}$ with $1 \le i \le n, 1 \le j \le m$:

$$\boxed{(\frac{\partial loss}{\partial S})_{i,j} = \frac{1}{4} g \lceil \frac{i}{2} \rceil \lceil \frac{j}{2} \rceil} \tag{3}$$

Since the problem already gives us the expression of S, which is:

$$S = \begin{bmatrix} s_{1,1} & s_{1,2} & \cdots & s_{1,2m} \\ s_{2,1} & s_{2,2} & \cdots & s_{2,2m} \\ \vdots & \vdots & \ddots & \vdots \\ s_{2n,1} & s_{2n,2} & \cdots & s_{2n,2m} \end{bmatrix} \tag{4}$$

For the Sigmoid Activation Layer where $S(C) = \frac{1}{1+e^{-C}}$, the gradient $\frac{\partial S}{\partial C}$ is (where * implies elementwise multiplication):

$$\frac{\partial S}{\partial C} = S * (1 - S) = \begin{bmatrix} s_{1,1}(1-s_{1,1}) & s_{1,2}(1-s_{1,2}) & \cdots & s_{1,2m}(1-s_{1,2m}) \\ s_{2,1}(1-s_{2,1}) & s_{2,2}(1-s_{2,2}) & \cdots & s_{2,2m}(1-s_{2,2m}) \\ \vdots & \vdots & \ddots & \vdots \\ s_{2n,1}(1-s_{2n,1}) & s_{2n,2}(1-s_{2n,2}) & \cdots & s_{2n,2m}(1-s_{2n,2m}) \end{bmatrix} \tag{5}$$

Thus, we can get $\frac{\partial loss}{\partial C}$ using chain rule:

$$\boxed{(\frac{\partial loss}{\partial C})_{i,j,d,e} = (\frac{\partial loss}{\partial S})_{i,j} (\frac{\partial S}{\partial C})_{d,e} = \frac{1}{4} g \lceil \frac{i}{2} \rceil \lceil \frac{j}{2} \rceil S_{d,e}(1 - S_{d,e})} \tag{6}$$

For the Convolution Layer, with the definition $C_{i,j} = \sum_h \sum_w K_{h,w} x_{i+h,j+w}$, we can get $\frac{\partial loss}{\partial K}$ using chain rule:

$$\boxed{(\frac{\partial loss}{\partial K})_{i,j,d,e,a,b} = (\frac{\partial loss}{\partial C})_{i,j,d,e} (\frac{\partial C}{\partial K})_{a,b} = \sum_{i,j} \frac{1}{4} g \lceil \frac{i}{2} \rceil \lceil \frac{j}{2} \rceil S_{d,e}(1 - S_{d,e}) X_{i+a,j+b}} \tag{7}$$

2 Problem 2: R-CNN

2.1 Please state at least two advantages of one-stage models compared with multi-stage models in object detection tasks.

Above all, the training process for **one-stage models** is more straight-forward and thus less expensive in space and time than **multi-stage models**.

Secondly, the training for **one-stage models** is a one-stage pipeline, thus the inner construction (i.e. the hidden layer) can communicate with each other better; while the training for **multi-stage models** is a multi-stage pipeline.

Finally, the **one-stage models** usually detects the target through regular and intensive sampling of position, scale and aspect ratio, which has the main advantage of computational efficiency; while for the **multi-stage models** (i.e. R-CNN), the object detection might be very slow.

2.2 Please state briefly why Non-maximum suppression(NMS) is generally needed for bounding-box based approaches in object detection tasks.

Non Maximum Suppression (NMS) is a class of algorithms to select one entity (e.g., bounding boxes) out of many overlapping entities. We can choose the selection criteria to arrive at the desired results. The criteria are most commonly some form of probability number and some form of overlap measure (e.g. Intersection over Union).

For bounding-box based approaches in object detection tasks, the neural network will generate multiple detection bounding-box, but most of the bounding-box contents are redundant, thus in order to find the solution an Non Maximum Suppression (NMS) will be generally needed to select the highest scores in the multiple testing bounding-box. And in this way, the total number of bounding-box will be greatly reduced and thus accelerating the training process.

3 Problem 3: CNN

3.1 Theoretical Analysis

3.1.1 Forward Propagation

$$Y = WX^T = \begin{bmatrix} 0.15 & 0.25 \\ 0.2 & 0.3 \end{bmatrix} \begin{bmatrix} 0.05 \\ 0.1 \end{bmatrix} = \begin{bmatrix} 0.0325 \\ 0.04 \end{bmatrix} \tag{8}$$

$$Z = sigmoid(Y) = \begin{bmatrix} 0.508124285 \\ 0.509998667 \end{bmatrix} \tag{9}$$

$$O = MZ = \begin{bmatrix} 0.40 & 0.50 \\ 0.45 & 0.55 \end{bmatrix} \begin{bmatrix} 0.508124285 \\ 0.509998667 \end{bmatrix} = \begin{bmatrix} 0.458249 \\ 0.509155 \end{bmatrix} \tag{10}$$

$$Q = sigmoid(O) = \begin{bmatrix} 0.6125987 \\ 0.6246084 \end{bmatrix} \tag{11}$$

$$E = 0.5[(0.01 - Q1)^2 + (0.99 - Q2)^2] = 0.248318 \tag{12}$$

3.1.2 Backward Propagation

$$\frac{\partial E}{\partial Q} = -(P - Q) = \begin{bmatrix} 0.6025987 \\ -0.3653916 \end{bmatrix} \tag{13}$$

$$\frac{\partial Q}{\partial O} = -Q(1 - Q) = \begin{bmatrix} 0.2373215 \\ 0.2344727 \end{bmatrix} \tag{14}$$

$$\frac{\partial O}{\partial M} = Z = \begin{bmatrix} 0.508124285 \\ 0.508124285 \end{bmatrix} \tag{15}$$

$$\frac{\partial O}{\partial Z} = M = \begin{bmatrix} 0.4 & \cdots \\ 0.45 & \cdots \end{bmatrix} \tag{16}$$

$$\frac{\partial Z}{\partial Y} = Z(1-Z) = \begin{bmatrix} 0.249933996 \\ \vdots \end{bmatrix} \tag{17}$$

$$\frac{\partial Y}{\partial W} = W = \begin{bmatrix} 0.15 & \cdots \\ \vdots & \ddots \end{bmatrix} \tag{18}$$

3.1.3 Theoretical Analysis Result

thus, we can compute the value of w11 and m11 as:

$$\boxed{m11 = m11 - \alpha\frac{\partial E}{\partial m11} = m11 - \alpha\frac{\partial E}{\partial Q_1}\frac{\partial Q_1}{\partial O_1}\frac{\partial O_1}{\partial m11} = 0.363667} \tag{19}$$

$$\boxed{w11 = w11 - \alpha\frac{\partial E}{\partial w11} = w11 - \alpha(\frac{\partial E}{\partial Q_1}\frac{\partial Q_1}{\partial O_1}\frac{\partial O_1}{\partial Z_1} + \frac{\partial E}{\partial Q_2}\frac{\partial Q_2}{\partial O_2}\frac{\partial O_2}{\partial Z_1})\frac{\partial Z_1}{\partial Y_1}\frac{\partial Y_1}{\partial w11} = 0.1496504} \tag{20}$$

3.2 Code Analysis

3.2.1 Code

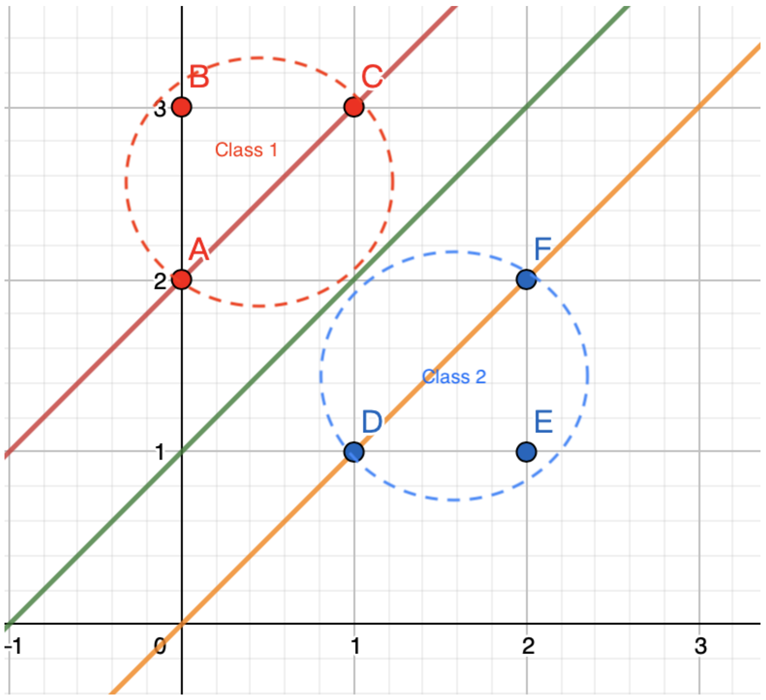
```
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def Dsigmoid(x):
    return sigmoid(x)*(1-sigmoid(x))

class CNN(object):
    def __init__(self, W, M, lr):
        self.lr = lr
        self.W = W
        self.M = M

    def forward(self, X, y):
        self.X = X
        self.Y = np.dot(self.W,X.T)
        self.Z = sigmoid(self.Y)
        self.O = np.dot(self.M, self.Z)
        self.Q = sigmoid(self.O)
        self.loss = np.sum((self.Q - y) * (self.Q - y).T)/ 2
        self.D = (self.Q - y) * Dsigmoid(self.O)

    def backward(self):
        dM = np.dot(self.Z.T, self.D)
        d1 = np.dot(self.D, self.M.T) * Dsigmoid(self.Y)
        dW = np.dot(self.X.T, d1)

        self.M -= self.lr * dM
        self.W -= self.lr * dW

if __name__ == '__main__':
    X = np.array([0.05,0.10])
    p = np.array([0.01,0.99])
    W = np.array([[0.15,0.25], [0.20,0.30]])
    M = np.array([[0.40,0.50], [0.45,0.55]])
    nn = CNN(W, M, lr=0.5)
    nn.forward(X, p)
    nn.backward()

    print(nn.W)
    print(nn.M)
```

3.2.2 Code Analysis Result

W = [[0.1496949 0.2496949] [0.1996949 0.2996949]]
M = [[0.36551357 0.46551357] [0.41551357 0.51551357]]
Thus, considering all above, we can conclude that:

$$\boxed{w11 = 0.1497, m11 = 0.3655} \tag{21}$$

4 Problem 4: SVM



According to lecture slides, since they reach the optimum at the same time, in order to find the maximum margin, we just need to minimize $\frac{1}{2}||w||^2$, where we have $||w|| = \sqrt{w_1^2 + w_2^2}$ due to L2 norm, s.t. $y(i)(w \cdot x(i) + b) \geq 1$, for i = 1,2,3,4,5,6. Since this is a binary classification problem, we need to find one separating hyperplane. Assume Class 1 as 1 (i.e. positive) and Class as -1 (i.e. negative), then we can get the following:

$$\begin{cases} w_1 \cdot 0 + w_2 \cdot 2 + b \geq 1 \\ w_1 \cdot 0 + w_2 \cdot 3 + b \geq 1 \\ w_1 \cdot 1 + w_2 \cdot 3 + b \geq 1 \\ -w_1 \cdot 1 - w_2 \cdot 1 - b \geq 1 \\ -w_1 \cdot 2 - w_2 \cdot 1 - b \geq 1 \\ -w_1 \cdot 2 - w_2 \cdot 2 - b \geq 1 \end{cases} \tag{22}$$

the equation above can be simplified as:

$$\begin{cases} 1 - w_2 \cdot 2 - b \leq 0 \\ 1 - w_2 \cdot 3 - b \leq 0 \\ 1 - w_1 \cdot 1 - w_2 \cdot 3 - b \leq 0 \\ 1 + w_1 \cdot 1 + w_2 \cdot 1 + b \leq 0 \\ 1 + w_1 \cdot 2 + w_2 \cdot 1 + b \leq 0 \\ 1 + w_1 \cdot 2 + w_2 \cdot 2 + b \leq 0 \end{cases} \tag{23}$$

Then, we can easily get the loss function shown as following:

$$\begin{aligned} Loss(w) = \frac{1}{2}(w_1^2 + w_2^2) + \beta_1(1 - 2w_2 - b) + \beta_2(1 - 3w_2 - b) + \beta_3(1 - w_1 - 3w_2 - b) \\ + \beta_4(1 + w_1 + w_2 + b) + \beta_5(1 + w_1 + 2w_2 + b) + \beta_6(1 + 2w_1 + 2w_2 + b) \end{aligned} \tag{24}$$

Then let's take the derivative w.r.t w_1,w_2 and b respectively and assign all of $\beta_1(1-2w_2-b), \beta_2(1-3w_2-b), \beta_3(1-w_1-3w_2-b), \beta_4(1+w_1+w_2+b), \beta_5(1+w_1+2w_2+b), \beta_6(1+2w_1+2w_2+b)$ to be zero since they are all smaller or equal to 0 from the proof shown above as equation (22) and (23), then we will have:

$$\begin{cases} \frac{\partial Loss}{\partial w_1} = w_1 - \beta_3 + \beta_4 + 2\beta_5 + 2\beta_6 = 0 \\ \frac{\partial Loss}{\partial w_2} = w_2 - 2\beta_1 - 3\beta_2 - 3\beta_3 + \beta_4 + \beta_5 + 2\beta_6 = 0 \\ \frac{\partial Loss}{\partial b} = -\beta_1 - \beta_2 - \beta_3 + \beta_4 + \beta_5 + \beta_6 = 0 \\ 1 - w_2 \cdot 2 - b = 0 \\ 1 - w_2 \cdot 3 - b = 0 \\ 1 - w_1 \cdot 1 - w_2 \cdot 3 - b = 0 \\ 1 + w_1 \cdot 1 + w_2 \cdot 1 + b = 0 \\ 1 + w_1 \cdot 2 + w_2 \cdot 1 + b = 0 \\ 1 + w_1 \cdot 2 + w_2 \cdot 2 + b = 0 \end{cases} \tag{25}$$

Solve the equation above, we will get:

$$\begin{cases} w_1 = -1 \\ w_2 = 1 \\ b = -1 \end{cases} \tag{26}$$

Plug back to the general equation above, we will get the expression of 2-D hyperplane:

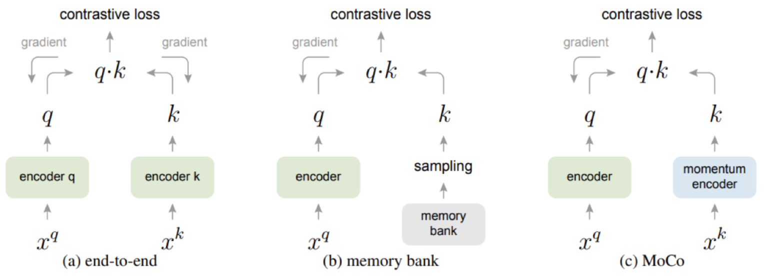
$$\boxed{-x_1 + x_2 - 1 = 0} \tag{27}$$

And from the figure above, we can easily figure out the support vectors to be the $\boxed{observation 1, 3, 4, 6}$.

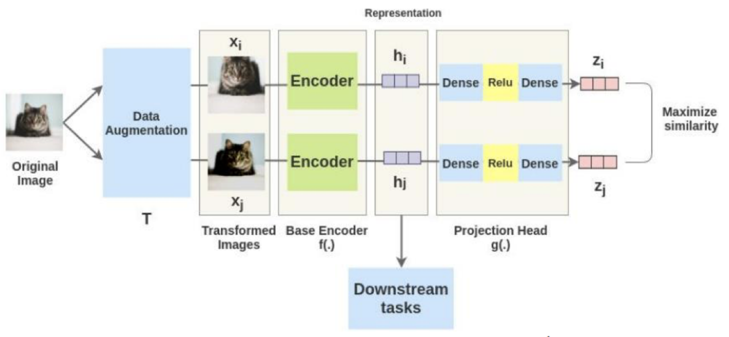
5 Problem 5: Self-supervised Learning

5.1 Describe the difference between SimCLR, MoCo, BYOL.

According to the original paper, **SimCLR** stands for "A Simple Framework for Contrastive Learning of Visual Representations", **MoCo** stands for "Momentum Constract for Unsupervised Visual Representation Learning" and **BYOL** stands for "Bootstrap Your Own Latent: A New Approach to Self-supervised Learning" and all of them are self-supervised constrastive learning methods.



MoCo actually uses a variant of Contrastive loss named InfoNCE, which uses a positive example and K negative examples (i.e. enlarge the set of negative samples), so that only the samples that really match (i.e. calculate the dot product with query q) are more similar, and when the unmatched ones are not similar at the same time, the loss is low. In this way, the encoder and autoregressive model can be optimized simultaneously. **MoCo** is a fusion version of end-to-end and memory bank model (i.e. use momentum encoder). It considers dictionary as a queue to maintain the current negative candidates pool, and it uses the dynamic update mechanism of the queue, that is after sampling each batch key, dequeue operations are performed to the keys corresponding to some of the relatively earliest mini-batches that entered the queue to ensure that some outdated and weakly consistent keys can be cleared.



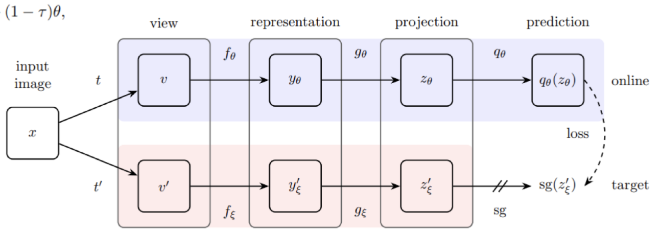
$$-\frac{1}{|\mathcal{X}_+|}\log\frac{\sum_{\mathbf{x}'\in\mathcal{X}_+}\exp\left(\mathrm{sim}(f_{\Theta}(\mathbf{x}), f_{\Theta}(\mathbf{x}'))/\tau\right)}{\sum_{\mathbf{x}'\in(\mathcal{X}_+\cup\mathcal{X}_-)}\exp\left(\mathrm{sim}(f_{\Theta}(\mathbf{x}), f_{\Theta}(\mathbf{x}'))/\tau\right)}$$

MoCo emphasizes that the number of pairs of samples is very important for comparative learning, and **SimCLR** believes that the way to construct negative examples is also very important. **SimCLR** samples some pictures (i.e. batch) and does different data augmentation on the image in the batch, and regard them as positive. It uses a basic neural network encoder, which extracts the representation vector from the enhanced data, and a small neural network projection head, which maps the representation to the space of contrast loss, hoping that the results of the same image but different augmentations are similar, and mutually exclusive of other results. And after training, the feature representation can be used downstream for fine-tuning (i.e. image classification).

$$\mathcal{L}_{\theta,\xi} \triangleq \left\|\overline{q_\theta}(z_\theta) - \bar{z}'_\xi\right\|_2^2 = 2 - 2\cdot\frac{\langle q_\theta(z_\theta), z'_\xi\rangle}{\left\|q_\theta(z_\theta)\right\|_2\cdot\left\|z'_\xi\right\|_2}$$

$$\theta \leftarrow \mathrm{optimizer}\left(\theta, \nabla_\theta\mathcal{L}^{\mathtt{BYOL}}_{\theta,\xi}, \eta\right),$$

$$\xi \leftarrow \tau\xi + (1-\tau)\theta,$$



BYOL believes that the previous methods are based on negative pairs, and they largely depend on the choice of image enhancement, so why not start directly from the perspective of image enhancement. **BYOL** uses no pair but two neural networks that interact and learn from each other, called the online network and the target network. The online network includes embedding, projection and prediction, which embeds the modules we want most, and the target network includes embedding and projection. The online network parameters are updated using the gradient of L2, and the target network is obtained directly from the online's momentum, where the target serves as the previous negative sample function.

According to my understanding, **MoCo** and **SimCLR** are more inclined to ask what is the difference between these two pictures while **BYOL** might be more likely to ask what is the difference between this picture and the average of these pictures.

6 Problem 6: RNN

The general idea and proof for vanishing and exploding gradient with both intuition and mathematics is shown as follows and we will discuss in detail after this part.

- Recall: $\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)$

- Therefore: $\dfrac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} = \operatorname{diag}\left(\sigma'\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right)\boldsymbol{W}_h$ (chain rule)

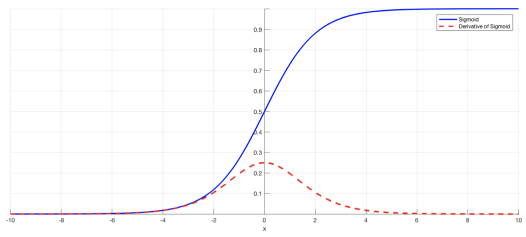
$$\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j<t\leq i} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} \qquad \text{(chain rule)}$$

$$= \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \boxed{\boldsymbol{W}_h^{(i-j)}} \prod_{j<t\leq i} \operatorname{diag}\left(\sigma'\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right) \qquad \text{(value of } \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}\text{)}$$

If W_h is small, then this term gets vanishingly small as *i* and *j* get further apart

$$\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j<t\leq i} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} \qquad \text{(chain rule)}$$

$$= \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \boxed{\boldsymbol{W}_h^{(i-j)}} \prod_{j<t\leq i} \boxed{\operatorname{diag}\left(\sigma'\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right)} \qquad \text{(value of } \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}\text{)}$$



- Value of sigmoid gradient is less than 0.25
- Multiple many times reach 0 quickly

- Consider matrix L2 norms:

$$\left\|\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}}\right\| \leq \left\|\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}}\right\| \left\|\boldsymbol{W}_h\right\|^{(i-j)} \prod_{j<t\leq i} \left\|\operatorname{diag}\left(\sigma'\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right)\right\|$$

- Pascanu et al showed that that if the largest eigenvalue of W_h is less than 1, then the gradient $\left\|\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}}\right\|$ will shrink exponentially
  - Here the bound is 1 because we have sigmoid nonlinearity

- There's a similar proof relating a largest eigenvalue >1 to exploding gradients

- Analyzing the norms of the Jacobians yields:

$$\left\|\frac{\partial h_j}{\partial h_{j-1}}\right\| \leq \|W^T\| \|\operatorname{diag}[f'(h_{j-1})]\| \leq \beta_W \beta_h$$

- Where we defined β's as upper bounds of the norms
- The gradient is a product of Jacobian matrices, each associated with a step in the forward computation.

$$\left\|\frac{\partial h_t}{\partial h_k}\right\| = \left\|\prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}\right\| \leq (\beta_W \beta_h)^{t-k}$$

- This can become very small or very large quickly [Bengio et al 1994], and the locality assumption of gradient descent breaks down. → **Vanishing or exploding gradient**

According to the lecture slides, we know that vanishing gradient problem happens when some terms in the chain rule are small since the gradient signal gets smaller and smaller as it back-propagates further. Conversely for the exploding gradient.

In this given case, we have the differentiate of Lt with respect to W shown as follows:

$$\frac{\partial L_t}{\partial W^o} = \sum_{t=0}^{T} \frac{\partial L_t}{\partial y_t} \frac{\partial y_t}{\partial W^o} \quad \ldots\ldots\ldots\ldots(1)$$

$$\frac{\partial L_t}{\partial W^i} = \sum_{t=0}^{T} \sum_{k=0}^{t} \frac{\partial L_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \left(\prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial W^i} \quad \ldots\ldots\ldots\ldots(2)$$

$$\frac{\partial L_t}{\partial W^h} = \sum_{t=0}^{T} \sum_{k=0}^{t} \frac{\partial L_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \left(\prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial W^h} \quad \ldots\ldots\ldots\ldots(3)$$

In the equation (2) and (3) above, the term $\prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$ is clearly problematic with the knowledge in the lecture slides.

On the one hand, if the largest eigenvalue of W_h is less than 1, then the gradient $||\frac{\partial h_j}{\partial h_{j-1}}||$ will shrink exponentially; On the other hand, if the largest eigenvalue of W_h is larger than 1, then the gradient $||\frac{\partial h_j}{\partial h_{j-1}}||$ will extend exponentially.

Since the gradient is a product of Jacobian matrices, each associated with a step in the forward computation, the term mentioned above can become very small or very large quickly, and the locality assumption of gradient descent breaks down, thus causing vanishing or exploding gradient.