# CIE6032 and MDS6232: Homework #2

Due on Wednesday, April 28th, 2021, 23:59pm

## Problem 1. CNN Equivariance [25 Points]

Equivariance is an appealing property when design neural network operations. It means that transforming the input image (e.g., translation) will also transform the output feature maps similarly after certain operations.

Formally, denote the image coordinate by $x \in \mathbb{Z}^2$, and the pixel values at each coordinate by a function $f: \mathbb{Z}^2 \to \mathbb{Z}^K$, where $K$ is the number of image channels. A convolution filter can also be formulated as a function $w: \mathbb{Z}^2 \to \mathbb{Z}^K$. Note that $f$ and $w$ are zero outside the image and filter kernel region, respectively. The convolution operation (correlation indeed for simplicity) is thus defined by

$$[f * w](x) = \sum_{\mathbf{y} \in \mathbb{Z}^2} \sum_{k=1}^{K} f_k(y) w_k(y - x). \tag{1}$$

a. **[5 Points]** Let $L_t$ be the translation $x \to x + t$ on the image or feature map, i.e., $[L_t f](x) = f(x - t)$. Prove that convolution has equivariance to translation:

$$[[L_t f] * w](x) = [L_t [f * w]](x), \tag{2}$$

which means that first translating the input image then doing the convolution is equivalent to first convolving with the image and then translating the output feature map. **(Hints: Use the formula (1) for the proof.)**

b. **[5 Points]** Let $L_{\mathbf{R}}$ be the $90^o$-rotation on the image or feature map, where

$$\mathbf{R} = \begin{bmatrix} \cos(\pi/2) & -\sin(\pi/2) \\ \sin(\pi/2) & \cos(\pi/2) \end{bmatrix}, \tag{3}$$

then $[L_{\mathbf{R}} f](x) = f(\mathbf{R}^{-1} x)$. However, convolution is not equivariant to rotations, i.e., $[L_{\mathbf{R}} f] * w \neq L_{\mathbf{R}} [f * w]$, which is illustrated by Figure 1 ((a) is not equivalent to (b) rotated by $90^o$). In order to establish the equivalence, the filter also needs to be rotated (i.e. (b) is equivalent to (c) in Figure 1). Prove that:

$$[[L_{\mathbf{R}} f] * w](x) = L_{\mathbf{R}} [f * [L_{\mathbf{R}^{-1}} w]](x) \tag{4}$$

**(Hints: Use the formula (1) for the proof.)**

To make convolution equivariant to rotations, we need to extend the definition of convolution and transformation. Recall a group $(G, \otimes)$ in algebra is a set $G$, together with an binary operation $\otimes$, which satisfies four requirements:

**Closure** $a \otimes b \in G, \forall a, b \in G$ .

**Associativity** $(a \otimes b) \otimes c = a \otimes (b \otimes c), \forall a, b, c \in G$ .

**Identity element** There exists a unique $e \in G, e \otimes a = a \otimes e = a, \forall a \in G$ .

**Inverse element** $\forall a \in G, \exists a^{-1} \in G, a \otimes a^{-1} = a^{-1} \otimes a = e$ .

We can formulate $90^{o}$-rotation and translation by a group $(G, \otimes)$ consisting of

$$g(r,u,v) = \begin{bmatrix} \cos(r\pi/2) & -\sin(r\pi/2) & u \\ \sin(r\pi/2) & \cos(r\pi/2) & v \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

where $r \in \{0,1,2,3\}$ and $(u,v) \in \mathbb{Z}^2$. $G = \{g\}$ and $\otimes$ is matrix multiplication. Translation is a special case of $G$ when $r = 0$ (i.e., $g(0,u,v)$ ) and rotation is a special case of $G$ when $u = v = 0$ (i.e., $g(r,0,0)$ ).

A key concept is to extend the definition of both the feature $f$ and the filter $w$ to $G$ . Imagine the feature map is duplicated four times with rotation of $0^{o}$, $90^{o}$, $180^{o}$, and $270^{o}$. Then $f(g)$ is the feature values at particular rotated pixel coordinate, and the convolution operation becomes

$$[f * w](g) = \sum_{\mathbf{h} \in G} \sum_{k=1}^{K} f_k(\mathbf{h}) w_k(g^{-1}\mathbf{h}). \tag{6}$$

A rotation-translation $u \in G$ on the feature map is thus $[L_u f](g) = f(u^{-1}g)$. Prove that under such extensions, the convolution is equivariant to rotation-translation:

$$[[L_u f] * w](g) = L_u [f * w](g). \tag{7}$$

c. **[15 Points]** Briefly explain how to implement this group convolution with traditional convolution and by rotating the feature map or filter.
**(Hints: Please read the paper "*Group Equivariant Convolutional Networks*").**

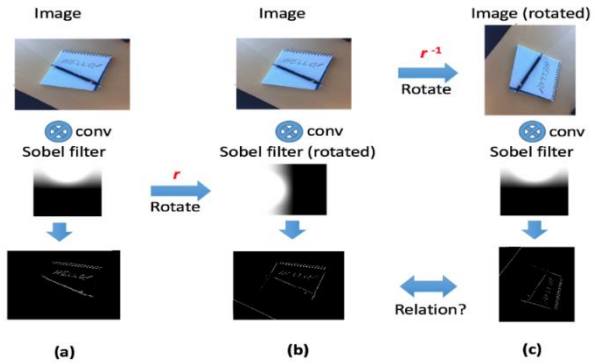


Figure 1: Equivariance relationship between convolution and rotation. (a) An image is convolved with a Sobel filer to detect horizontal edges. (b) The filter is rotated counterclockwise and then convolves the original image. (c) The image is first rotated clockwise, then it is convolved with the filter.

## Problem 2. RNN Backpropagation [10 points]

A recurrent neural network is shown in Figure 2,

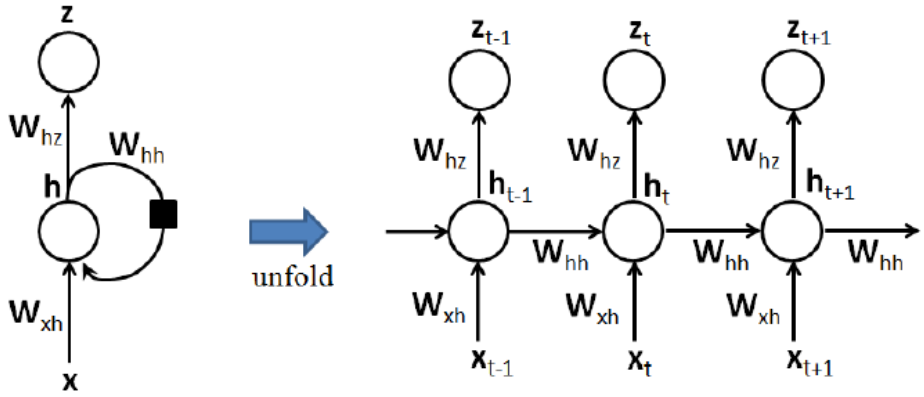


Figure 2: RNN fold and unfold structure.

$$h_t = \tanh(\mathbf{W}_{xh} x_t + \mathbf{W}_{hh} h_{t-1} + b_h)$$

$$z_t = \text{softmax}(\mathbf{W}_{hz} h_t + b_z)$$

The total loss for a given input/target sequence pair (x, y), measured in cross entropy

$$L(x, y) = \sum_t L_t = \sum_t -\log z_{t,y_t}$$

In the lecture, we provide the general idea of how to calculate the gradients $\frac{\partial L}{\partial W_{hz}}$ and $\frac{\partial L}{\partial W_{hh}}$.

Please provide the details of the algorithms and equations, considering the mapping and cost functions provided above.

## Problem 3. Q-Learning [25 points]

Consider an episodic, deterministic chain MDP with n = 7 states assembled in a line.

The possible actions are $a \in \{-1, 1\}$, and the transition function is deterministic such that $s' = s + a$. Note that as an exception, taking $a = -1$ from $s = 1$ keeps us in $s = 1$, and taking $a = -1$ from $s = 7$ keeps us in $s = 7$.

We have a special goal state, $g = 4$, such that taking any action **from** $g$ ends the episode with a reward of $r = 0$. From all other states, any action incurs a reward of $r = -1$. We let discount factor $\gamma = 1/3$.

The chain MDP is pictured in Figure 3, with the goal state $s_4$ shaded in.

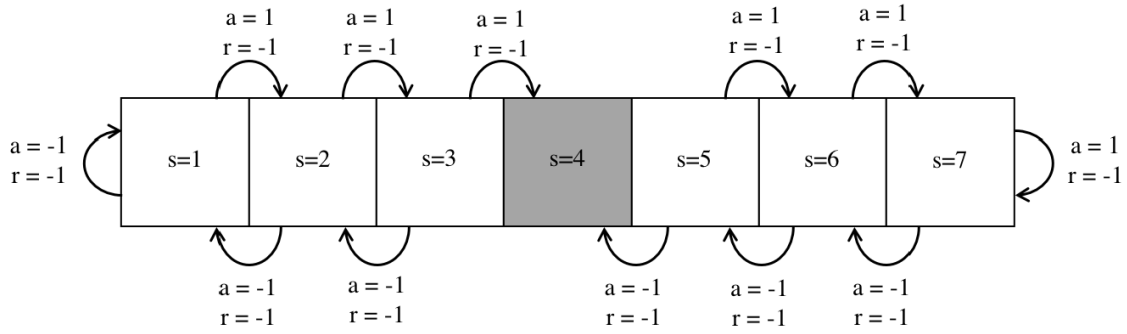


Figure 3. MDP procedure with seven states.

By inspection, we see that $V^* = -|s-4|$

a. **[5 Points]** We would like to perform tabular **Q-learning** on this chain MDP. Suppose we observe the following 4 step trajectory (in the form (*state, action, reward*)):

$$(3,\ -1,\ -1),\ (2,\ 1,\ -1),\ (3,\ 1,\ -1),\ (4,\ 1,\ 0)$$

Suppose we initialize all Q values to 0. Use the tabular Q-learning update to give updated values for

$$Q(3,\ -1),\ Q(2,\ 1),\ Q(3,\ 1)$$

assuming we process the trajectory in the order given from left to right. Use the learning rate $\alpha = 1/2$.

[**Hints: Using** $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a' \in \{1,-1\}} Q(s',a') - Q(s,a))$ ]

Now, we are interested in performing linear function approximation in conjunction with Q-learning. In particular, we have a weight vector $w = [w_0, w_1, w_2]^T \in \mathbb{R}^3$. Given some state $s$ and action $a \in \{-1,1\}$, the featurization of this state, action pair is: $[s,a,1]^T$. To approximate the Q-values, we compute

$$\hat{q}(s,a;w) = [w_0, w_1, w_2][s,a,1]^T = w_0 * s + w_1 * a + w_2$$

Given the parameters $w$ and a single sample $(s,a,r,s')$. The loss function we will minimize is

$$J(w) = (r + \gamma \max_{a'} \hat{q}(s',a';w^-) - \hat{q}(s,a;w))^2,$$

where $\hat{q}(s',a';w^-)$ is a target network parameterized by fixed weights $w^-$.

b. **[15 Points]** Suppose we currently have a weight vectors $w = [-1,1,1]^T$ and $w^- = [1,-1,-2]^T$, and we observe a sample $(s=2, a=-1, r=-1, s'=1)$.

Perform a single gradient update to the parameters $w$ given this sample. Use the learning rate $\alpha = 1/4$. Write out the gradient $\nabla_w J(w)$ as well as the new parameters $w'$.

c. **[5 Points]** The optimal Q function $Q^*(s, a)$ is exactly representable by some neural network architecture $N$. Suppose we perform Q-learning on this MDP using the architecture $N$ to represent the Q-values. Suppose we randomly initialize the weights of a neural net with architecture $N$ and collect infinitely many samples using an exploration strategy that is greedy in the limit of infinite exploration (GLIE).
Are we guaranteed to converge to the optimal Q function $Q^*(s, a)$? Explain it.

# Problem 4. Coding [40 points]

In this problem we will implement a convolutional neural network to solve the problem of image classification on the CIFAR-10 dataset.

The network should consist of two convolutional layers and two fully connected layers. The first convolution has a kernel size of 5, stride of 1 and outputs 6 channels (without padding); the second convolution shares the same kernel setting and outputs 16 channels. The two fully connected layers have the number of output neurons as 120, 84, respectively. The last layer before the loss is an additional FC layer with ten outputs (the number of object classes on CIFAR-10). Max pooling with stride of 2 is appended after each convolution. And we use ReLU as the activation function throughout the network. Note that you may optionally add the dropout layer after each fully connected layer, depending on the training behavior of your network. The loss is a softmax loss. Other training specifications (recommended and yet not guaranteed): we use SGD as the optimization method; the base learning rate is 0.001. Momentum and weight decay are set to be 0.9 and 0.0005. The learning rate policy is step, which is the way of reducing learning rates. You can refer to AlexNet paper or AlexNet tutorial for details. The initial weights of filters are Gaussian distributed with zero mean and 0.01 std. You do not need to worry about these settings. They are already set by default in the code we provide.

The starter code can be found in *coding/simple_cifar.py*, which includes the general framework, dataset loading and evaluation process.

a. **[10 points]** Set up the network architecture specified above. Draw the training curve and test accuracy for at least 20 epochs. Visualize the filters learned in the first convolutional layer. Report and analyze the performance improvement of using softmax loss and regularization.

b. **[10 points] Weight initialization.**

A good initialization of the network should ensure the variance of signals among layers constant. For a convolution or fully connected layer, we have

$$y_l = \mathbf{W}_l x_l + b_l, \qquad (8)$$

where $\mathrm{y}_l$, $\mathrm{x}_l$ are the output and input maps of layer $l$, $\mathrm{W}_l$, $\mathrm{b}_l$ denote the weights and bias of the filter. We also have $\mathrm{x}_l = f\left(\mathrm{y}_{l-1}\right)$ where $f$ is the activation function (we use ReLU throughout the discussion). Let $y_l$, $x_l$, $w_l$ represent the random variables of each element in $\mathrm{y}_l$, $\mathrm{x}_l$, $\mathbf{W}_l$, respectively. Assume that $w_l$ has zero mean and elements in $\mathrm{x}_l$, $\mathbf{W}_l$ are mutually independent and these two are also independent. Then we have:

$$Var[y_l] = n_l Var[w_l x_l], \tag{9}$$

where $n_l$ is the number of neurons in layer $l$ . In case you are not familiar with the conclusion above, there is a blog. Verify the following:

$$Var[y_l] = n_l Var[w_l] E[x_l^2] \tag{10}$$

$$E[x_l^2] = \frac{1}{2} Var[y_{l-1}]. \tag{11}$$

Putting Eqn. 11 back into Eqn. 10 and considering all layers from 1 to L, we have derived the key to the initialization design:

$$Var[y_L] = Var[y_1]\left(\prod_{l=2}^{L} \frac{1}{2} n_l Var[w_l]\right). \tag{12}$$

A good initialization method should avoid reducing or magnifying the magnitudes of input signals exponentially. A sufficient condition from Eqn. 12 is thereby:

$$\frac{1}{2} n_l Var[w_l] = 1, \quad \forall l \tag{13}$$

From above, we derive a proper way of initializing the weights in the network. Similar conclusion holds for a backpropagation case (see paper for details).

In this question, you need to do two things.

(b1) Prove Eqn. 10 and Eqn. 11.

(b2) Now based on the network designed previously, ***implement such an idea*** to initialize weights in your network and verify if such a design works. Report the test accuracy compared with the result in question a.

c. **[10 points] Batch normalization**.

We have discussed the technique of batch normalization in the lecture. Add BN layer (using existing library in Pytorch is acceptable, but implementing a BN layer by

yourself is more preferable) after each convolution and FC layer (except the very last FC for classification) and verify the effectiveness of BN via test accuracy.

d. **[10 points] Try other ideas.**
For example, investigate the effect of data augmentation (rotation, scaling, etc.); different optimization policies (Adam, RMSProp, etc.), and other normalization methods. Note that for each new component, you have to write your own code/layer instead of borrowing from the existing library. And some brief analysis should also be appended.

**[Note:]**

- Please write a CIFAR_A2.ipynb file including all your code/figure/necessary (and brief).
- Submit your notebook file as a single file. NOTE that the derivation of theory part in Problem 4 has to be written alongside with other problems; no need to write it again in the notebook.
- In python notebook, you cannot parse arguments via argparse. Just use fixed (and well-tuned) parameters in your submission, removing the parser part in the simple_cifar.py.
- The loss in the demo may be misleading from the cross entropy loss. Think about this and decide which one is the correct version.