CIE6032 and MDS6232: Homework 2

Jian Zhan, 219012058

April 18, 2020

1. **CNN Equivariance**

   a. Prove that convolution has equivariance to translation.

$$\begin{aligned}
[[L_t f] \star w](x) &= \sum_y f(y-t)w(y-x) \\
&= \sum_y f(y)w(y+t-x) \\
&= \sum_y f(y)w(y-(x-t)) \\
&= [L_t[f \star w]](x)
\end{aligned}$$

   b. Prove that convolution is not equivariant to rotations.

$$\begin{aligned}
[[L_r f] \star w](x) &= \sum_y \sum_k L_r f_k(y) w_k(y-x) \\
&= \sum_y \sum_k f_k\left(r^{-1}y\right) w_k(y-x) \\
&= \sum_y \sum_k f_k(y) w_k(ry-x) \\
&= \sum_y \sum_k f_k(y) w_k\left(r\left(y-r^{-1}x\right)\right) \\
&= \sum_y \sum_k f_k(y) L_{r^{-1}} w\left(y-r^{-1}x\right) \\
&= f \star [L_{r^{-1}} w]\left(r^{-1}x\right) \\
&= L_r\left[f \star [L_{r^{-1}} w]\right](x)
\end{aligned}$$

   To make convolution equivariant to rotations, we need to extend the definition of convolution and transformation. Prove that under the specific extensions, the convolution is equivariant to rotation-translation.

$$
\begin{aligned}
[[L_u f] \star w](g) &= \sum_{h \in G} \sum_k f_k \left(u^{-1} h\right) w \left(g^{-1} h\right) \\
&= \sum_{h \in G} \sum_k f(h) w \left(g^{-1} u h\right) \\
&= \sum_{h \in G} \sum_k f(h) w \left(\left(u^{-1} g\right)^{-1} h\right) \\
&= [L_u [f \star w]](g)
\end{aligned}
$$

c. Briefly explain how to implement this group convolution with traditional convolution and by rotating the feature map or filter.

For G-Equivariant correlation, the inputs f and w are functions on Z^2, so $L_u f$ denotes the transformation of such a function, while $L_u[f \star w]$ denotes the transformation of the feature map, which is a function on G.

The implementation is essentially straightforward, for any transformation, it can be changed into the form of groups G by definition and apply the formula afterwards.

2. **RNN Backpropagation**

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

Algorithm steps:

(a) A single time step of the input is supplied to the network i.e. x_t is supplied to the network

(b) We then calculate its current state using a combination of the current input and the previous state i.e. we calculate h_t

(c) The current h_t becomes h_{t-1} for the next time step

(d) We can go as many time steps as the problem demands and combine the information from all the previous states

(e) Once all the time steps are completed the final current state is used to calculate the output y_t

(f) The output is then compared to the actual output and the error is generated

(g) The error is then backpropagated to the network to update the weights(we shall go into the details of backpropagation in further sections) and the network is trained

In this part, we have equations:

$$
\begin{aligned}
h_t &= \tanh\left(\mathbf{W}_{xh} x_t + \mathbf{W}_{hh} h_{t-1} + b_h\right) \\
z_t &= \operatorname{softmax}\left(\mathbf{W}_{hz} h_t + b_z\right)
\end{aligned}
$$

Then, we discuss more on backpropagation.

The loss(error) is given by equation:

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

The steps for backpropagation can be summarized:

(a) The cross entropy error is first computed using the current output and the actual output

(b) Remember that the network is unrolled for all the time steps

(c) For the unrolled network, the gradient is calculated for each time step with respect to the weight parameter

(d) Now that the weight is the same for all the time steps the gradients can be combined together for all time steps

(e) The weights are then updated for both recurrent neuron and the dense layers

In this part, we have equations on gradients:

$$\frac{\partial L}{\partial L_t} = 1, \quad \frac{\partial L}{\partial \mathbf{z}_t} = \frac{\partial L}{\partial L_t}\frac{\partial L_t}{\partial \mathbf{z}_t} = \frac{\partial L_t}{\partial \mathbf{z}_t}$$

$$\frac{\partial L}{\partial \mathbf{W}_{hz}} = \sum_t \frac{\partial L_t}{\partial \mathbf{z}_t}\frac{\partial \mathbf{z}_t}{\partial \mathbf{W}_{hz}}, \quad \frac{\partial L}{\partial \mathbf{b}_z} = \sum_t \frac{\partial L_t}{\partial \mathbf{z}_t}\frac{\partial \mathbf{z}_t}{\partial \mathbf{b}_z} \quad \frac{\partial L}{\partial \mathbf{W}_{hh}} = \sum_t \frac{\partial L}{\partial \mathbf{h}_t}\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_{hh}}$$

$$\frac{\partial L}{\partial \mathbf{h}_t} = \frac{\partial L}{\partial \mathbf{h}_{t+1}}\frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t} + \frac{\partial L}{\partial \mathbf{z}_t}\frac{\partial \mathbf{z}_t}{\partial \mathbf{h}_t}$$

3. **Q-Learning**

a. Use the tabular Q-learning update to give updated values for Q(3,-1), Q(2,1), Q(3,1)

With the hint equation, and $\alpha = 1/2, \gamma = 1$

$$Q(3,-1) \leftarrow 0 + \frac{1}{2}\left(-1 + \max_{a'} Q\left(2, a'\right)\right) = \frac{1}{2}(-1+0) = -\frac{1}{2}$$
$$Q(2,1) \leftarrow 0 + \frac{1}{2}\left(-1 + \max_{a'} Q\left(3, a'\right)\right) = \frac{1}{2}(-1+0) = -\frac{1}{2}$$
$$Q(3,1) \leftarrow 0 + \frac{1}{2}\left(-1 + \max_{a'} Q\left(4, a'\right)\right) = \frac{1}{2}(-1+0) = -\frac{1}{2}$$

b. 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'

$$\begin{aligned}\nabla_w J(w) &= -2\left(r + \gamma \max_{a'} \hat{q}\left(s', a'; w^-\right) - \hat{q}(s, a; w)\right) \nabla_w \hat{q}(s, a; w) \\ &= -2\left(r + \max_{a'} \left(w^-\right)^T \begin{bmatrix} s' \\ a' \\ 1 \end{bmatrix} - w^T \begin{bmatrix} s \\ a \\ 1 \end{bmatrix}\right) \begin{bmatrix} s \\ a \\ 1 \end{bmatrix}\end{aligned}$$

$$\begin{aligned}w' &\leftarrow w - \alpha \nabla_w J(w) \\ &= w + \frac{1}{2}\left(r + \max_{a'} \left(w^-\right)^T \begin{bmatrix} s' \\ a' \\ 1 \end{bmatrix} - w^T \begin{bmatrix} s \\ a \\ 1 \end{bmatrix}\right) \begin{bmatrix} s \\ a \\ 1 \end{bmatrix}\end{aligned}$$

For the sample (2, -1, -1, 1), with the equation, we have:

$$w' \leftarrow \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix} + \frac{1}{2}\left(-1 + \max_{a'} \begin{bmatrix} 1 \\ -1 \\ -2 \end{bmatrix}^T \begin{bmatrix} 1 \\ a' \\ 1 \end{bmatrix} - \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix}^T \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}\right) \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1/2 \\ 3/2 \end{bmatrix}$$

c. Are we guaranteed to converge to the optimal Q function $Q^*(s, a)$? Explain it.

We are not guaranteed to converge to the function, it is because this method involves function approximation, bootstrapping, and off-policy training.

4. **Coding**

a. Report and analyze the performance improvement of using softmax loss and regularization

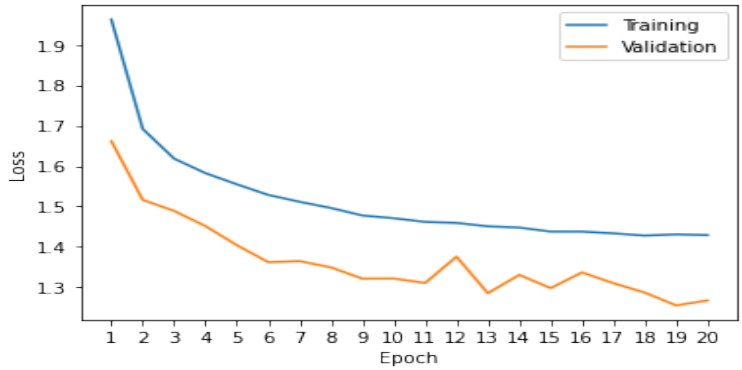

Figure 1: Training and Validation Loss

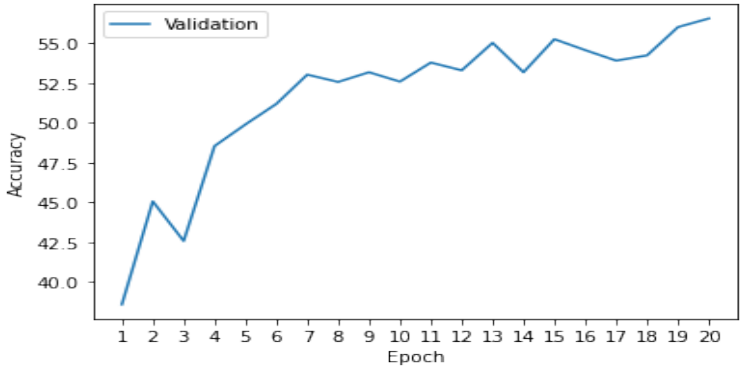

Figure 2: Test accuracy

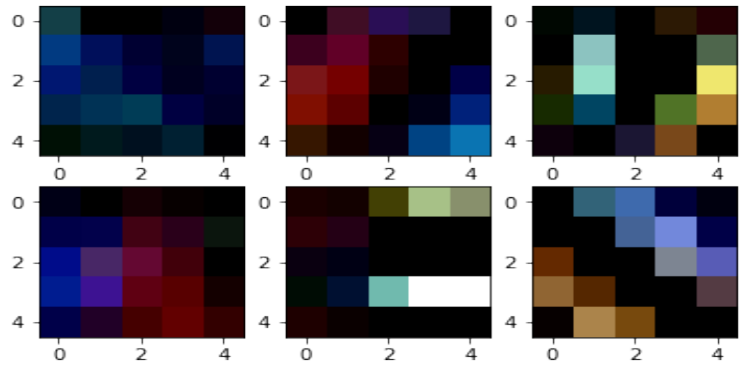

Figure 3: First Conv Filter Visualization

b. Weight initialization

(b1) Proof

Since w_l and x_l are independent of each other, and w_l has zero mean.

$$\begin{aligned}\operatorname{Var}[y_l] &= n_l \operatorname{Var}[w_l x_l] \\ &= n_l \operatorname{Var}[w_l] E\left[x_l^2\right]\end{aligned}$$

If we let w_{l-1} have a symmetric distribution around zero and $b_{l-1} = 0$, then y_{l-1} has zero mean and has a symmetric distribution around zero. This leads to $E\left[x_l^2\right] = \frac{1}{2}\operatorname{Var}[y_{l-1}]$ when f is ReLU.

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

Compare to the test accuracy in a, we can notice that the test accuracy is higher at the very begining. Also, we can reach a better test accuracy after 20 epochs.

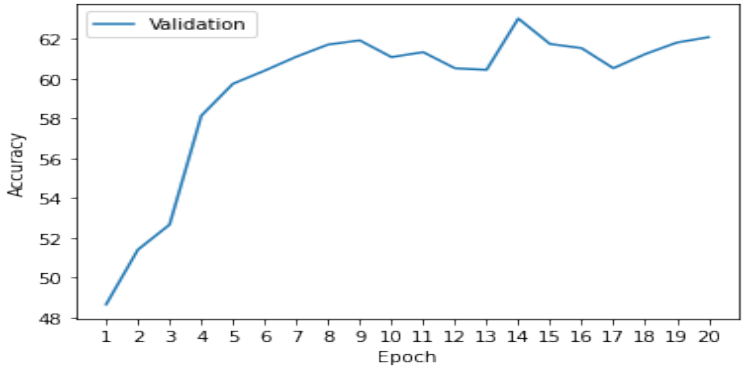

Figure 4: Test accuracy with Weight Initialization

c. Verify the effectiveness of BN via test accuracy.

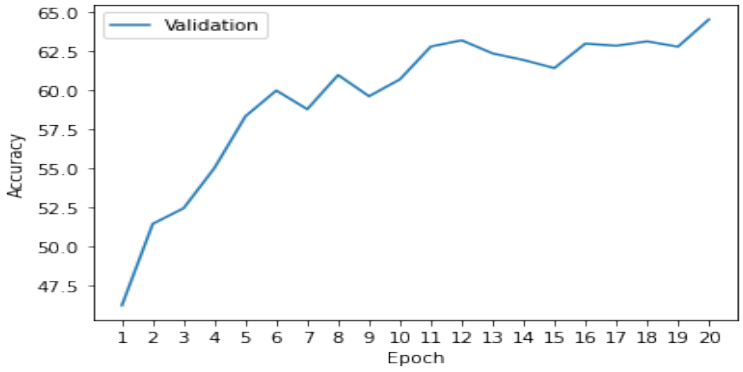

Figure 5: Test accuracy with BN

d. Try other ideas.

In my experiment, I have tried following ideas (some are commented in code part):

- Data Augmentation, using RandomHorizontalFlip and RandomCrop Transfroms
- Adam optimizer
- Different normalization methods, including GroupNorm/InstanceNorm/LayerNorm
- Dropout

However, these ideas do not help to improve the test accuracy. I think it is because the network I apply in this project is very simple one. Most of these methods essentially help to handle the overfit. If we build a much deeper network such as VGG16, ResNet18 and DPN92, then these methods may help to improve the performance of models.