Weekly Report

SHAO YUMING[1]

[1] *Email:1159326041@sjtu.edu.cn*

Compiled March 17, 2019

Because the previous week is much too busy , I failed to hand in a weekly report . All the work I have done during these three weeks can be found here . I finished all of the CS231n labs within these 3 weeks without falling behind my required courses to prove my bravery and perseverance , and preparing for the approaching mini-project as well .

1. DISCUSSION WITH PROF.MA AND PROF.SONG

On the Wednesday last week , I had a talk with Prof.Ma and Prof.Song . They showed me an opportunity of participating in a project concerning both CV and Computer System (But not having more information on that right now). Since I have finially finished CSAPP , CS299 and CS231n , I think I myself shall be a candidate ? :) If so , I will spare no effort to work hard for it as well but that should come behind my personal mini-project. :/

However , I was all strictly criticized by Prof.Song for my lack of weekly reports . I think he is right . But the fact is , I am just learning all the time , reading books and coding . I always find it hard to record something . For instance , should I write down the knowledge mentioned in the book chapters ? From my perspective , of course it's simply a waste of time .

But once I step into the phase of doing experiment , things will change a lot . The weekly reports listed in AISIG-Forum are good samples for me . Students record what experiment they are doing , methods they came up with and results they got . I think those issues are more easy and practical to write with .

As a conclusion , writing a weekly report for experiment is much easier than for reading a book (especially CSAPP with more than 1000 pages) .

2. CS231N

I have uploaded my solution to my github , here is the link . To make it lighter and easier , I just submitted all the edited part of the labs , something else such as utilities and pictures are not included . Here of course I won't describe the details on every step .

A. assignment 1

This is an easy task to solve , but I think some essential concepts have been involved in it . Due to the benefits of writing with Python , object-oriented programming method play a crucial role . I saw classifiers are defined as classes , with all the corresponding methods embedded , such as initialization , training , loss function , prediction . While writing my own ones later on , I shall act like this .

What's more , it is my first time to try to modify parameters by writing code iterating through given parameters and choosing the best ones with the highest prediction accuracy .

B. assignment 2

I think assignment 2 is much harder for more concepts and models not mentioned in CS229 . I spent a lot of time trying to generate mathematical expressions of matrix operation on myself and debug . Luckily I successfully passed every test . I feel that even some subtle problems can cause many troubles when the results are ridiculous and I am pushed to go back wondering what's wrong .

In this assignment , I was required to write forward and backward method for a tiny convolutional neural network . However what I can do is just to satisfy the requirement and implemented a naive version with many *for* loops .

```
out = None
###########################################################################
# TODO: Implement the convolutional forward pass.                         #
# Hint: you can use the function np.pad for padding.                      #
###########################################################################
N, _, H, W = x.shape
F, C, HH, WW = w.shape
S, P = conv_param['stride'], conv_param['pad']
H1 = int(1 + (H + 2 * P - HH) / S)
W1 = int(1 + (W + 2 * P - WW) / S)
out = np.zeros((N, F, H1, W1))
container = np.zeros((1, C, H + 2 * P, W + 2 * P))
for i in range(N):
    container[:, :, P:H + P, P:W + P] = x[i, :, :, :]
    for f in range(F):
        for r in range(H1):
            for c in range(W1):
                out[i, f, r, c] += np.sum(container[:, :, r * S:r * S + HH, c * S:c * S + WW] *
                                          w[f, :, :, :]) + b[f]
###########################################################################
#                             END OF YOUR CODE                            #
###########################################################################
cache = (x, w, b, conv_param)
return out, cache
```

Fig 1 : conv forward naive

And now I am just wondering , how to make a convolutional neural network implementation vectorized to make parallelism practical and speed up our training procedure . Because the sad fact is , the fast implementation provided by professors of CS231n written in Cython can be hundreds of times faster than my naive implementation .

Another thing to mention is that , while learning per-parameter adaptive learning rate methods , concretely algorithm Adagrad , RMSprop , Adam , my solution always has an error ratio of $1e-1$. After checking it out , I discovered that the term of *epsilon* is missed in a denominator . (epsilon can be found in the figure below)

```
def rmsprop(w, dw, config=None):
    """
    Uses the RMSProp update rule, which uses a moving average of squared
    gradient values to set adaptive per-parameter learning rates.

    config format:
    - learning_rate: Scalar learning rate.
    - decay_rate: Scalar between 0 and 1 giving the decay rate for the squared
      gradient cache.
    - epsilon: Small scalar used for smoothing to avoid dividing by zero.
    - cache: Moving average of second moments of gradients.
    """
    if config is None: config = {}
    config.setdefault('learning_rate', 1e-2)
    config.setdefault('decay_rate', 0.99)
    config.setdefault('epsilon', 1e-8)
    config.setdefault('cache', np.zeros_like(w))

    next_w = None
    ###########################################################################
    # TODO: Implement the RMSprop update formula, storing the next value of w #
    # in the next_w variable. Don't forget to update cache value stored in    #
    # config['cache'].                                                        #
    ###########################################################################
    cache = config['cache']
    decay_rate = config['decay_rate']
    epsilon = config['epsilon']
    learning_rate = config['learning_rate']
    cache = decay_rate * cache + (1 - decay_rate) * dw ** 2
    next_w = w - learning_rate * dw / (np.sqrt(cache) + epsilon)
    config['cache'] = cache
    ###########################################################################
    #                             END OF YOUR CODE                            #
    ###########################################################################

    return next_w, config
```

Fig 2 : rmsprop

After adding an epsilon at a correct position , I got a right answer . I think this phenomenon explain the importance of epsilon here . While other terms in the denominator is relatively small , we will accidentally encounter an error similar to divide-by-zero . The role epsilon plays here is to avoid this possibility .

C. assignment 3

This assignment is implemented with Pytorch instead of Tensorflow , because of my limited time . The matrix operation in LSTM implementation was extremely hard to generate by hand which cost me much time .

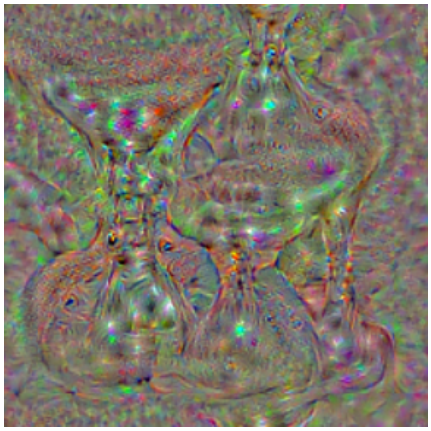

Fig 3 : hourglass generated by trained model

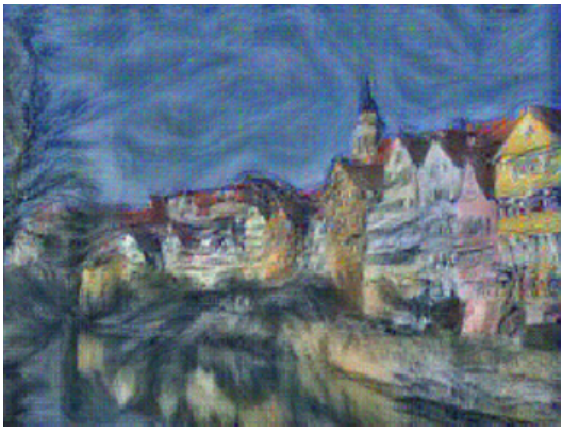

Fig 4 : a town in the style of masterpiece Starry Sky

The problem is , in the part of GANs , I do not quite understand :

```
def bce_loss(input, target):
    """
    Numerically stable version of the binary cross-entropy
        loss function.

    Inputs:
    - input: PyTorch Tensor of shape (N, ) giving scores.
    - target: PyTorch Tensor of shape (N,) containing 0
        and 1 giving targets.

    Returns:
    - A PyTorch Tensor containing the mean BCE loss over
        the minibatch of input data.
    """
    neg_abs = - input.abs()
    loss = input.clamp(min=0) - input * target + (1 +
        neg_abs.exp()).log()
    return loss.mean()


def discriminator_loss(logits_real, logits_fake):
    """
    Computes the discriminator loss described above.

    Inputs:
    - logits_real: PyTorch Tensor of shape (N,) giving
        scores for the real data.
    - logits_fake: PyTorch Tensor of shape (N,) giving
        scores for the fake data.

    Returns:
    - loss: PyTorch Tensor containing (scalar) the loss
        for the discriminator.
    """
    label1=(logits_real>0).type(dtype)
    label2=(logits_fake<0).type(dtype)
    loss=bce_loss(logits_real.abs(),label1)+
    bce_loss(logits_fake.abs(),label2)

    return loss

def generator_loss(logits_fake):
    """
    Computes the generator loss described above.

    Inputs:
    - logits_fake: PyTorch Tensor of shape (N,) giving
        scores for the fake data.

    Returns:
    - loss: PyTorch Tensor containing the (scalar) loss
```

```
    for the generator.
"""
label=(logits_fake>0).type(dtype)
loss=bce_loss(logits_fake.abs(),label)
return loss
```

Here is the code I tried which can pass the test with low error ratio . But I just don't know why it is correct implementation for the loss of discriminator and generator (code of bce loss is provided , no problem). And the function for computing the losses are as follows : the generator loss

$$L_G = -E_{z\sim p(z)}[\log D(G(z))] \tag{S1}$$

the discriminator loss

$$L_D = -E_{x\sim P_{data}}[\log D(x)] - E_{z\sim p(z)}[\log(1 - D(G(z)))] \tag{S2}$$

For more detail , the referrence is paper and my ipynb file.

3. CONCLUSION

Sincerely speaking , I did not pay enough attention to the utility parts of CS231n assignment code which is unnecessary for me to do the lab but quite crucial when writing code myself later . I will go back and check them when I have to .

I don't have any plan now . I am just waiting for my instructors' advice and do my homework of the required courses .