B093040024
郭晏涵

深度視覺

HW5：Neural Network

notebook 執行過程

依照說明，forward 代表實現該函數，backward 代表計算 gradient

實現 layer 中的 Sigmoid 函數

Task: Implement

Open the file `exercise_code/networks/layer.py`. Implement the `forward` and the `backward` method in the `Sigmoid` class, and test your implementation by running the following cell.

```
[5] # Test your sigmoid implementation
    print(SigmoidTestWrapper()())

    SigmoidForwardTest passed.
    SigmoidBackwardTest passed.
    Congratulations! You have passed all the unit tests!!! Tests passed: 2/2
    Score: 100/100
    You secured a score of: 100
```

```
    def forward(self, x):
        """
        :param x: Inputs, of any shape

        :return outputs: Outputs, of the same shape as x
        :return cache: Cache, stored for backward computation, of the same shape as x
        """
        shape = x.shape
        outputs, cache = np.zeros(shape), np.zeros(shape)
        ########################################################################
        # TODO:
        # Implement the forward pass of Sigmoid activation function
        ########################################################################

        outputs = 1 / (1 + np.exp(-x))
        cache = (x, outputs)

        ########################################################################
        #                           END OF YOUR CODE
        ########################################################################
        return outputs, cache
```

```
    def backward(self, dout, cache):
        """
        :return: dx: the gradient w.r.t. input X, of the same shape as X
        """
        dx = None
        ########################################################################
        # TODO:
        # Implement the backward pass of Sigmoid activation function
        ########################################################################

        dx = cache[1] * (1-cache[1]) * dout

        ########################################################################
        #                           END OF YOUR CODE
        ########################################################################
        return dx
```

實現 layers 中的 ReLU 函數(max(0, x))

Task: Implement

Open the file `exercise_code/networks/layer.py`. Implement the `forward` and the `backward` method in the `Relu` class, and test your implementation by running the following cell.

```
[6]  # Test your ReLu implementation
     print(ReluTestWrapper()())
```

```
ReluForwardTest passed.
ReluBackwardTest passed.
Congratulations! You have passed all the unit tests!!! Tests passed: 2/2
Score: 100/100
You secured a score of: 100
```

```python
    def forward(self, x):
        """
        :param x: Inputs, of any shape

        :return out: Outputs, of the same shape as x
        :return cache: Cache, stored for backward computation, of the same shape as x
        """
        outputs = None
        cache = None
        ########################################################################
        # TODO:
        # Implement the forward pass of Relu activation function               #
        ########################################################################

        outputs = np.array(list(map(lambda x: x if x > 0 else 0, x.ravel()))).reshape(x.shape)
        cache = (x, outputs)

        ########################################################################
        #                           END OF YOUR CODE
        ########################################################################
        return outputs, cache
```

```python
    def backward(self, dout, cache):
        """
        :return: dx: the gradient w.r.t. input X, of the same shape as X
        """
        dx = None
        ########################################################################
        # TODO:
        # Implement the backward pass of Relu activation function              #
        ########################################################################

        dx = np.array(list(map(lambda x: 1 if x > 0 else 0, cache[1].ravel()))).reshape(cache[1].shape) * dout

        ########################################################################
        #                           END OF YOUR CODE
        ########################################################################
        return dx
```

實作 layer 中的 affine_forward、affine_backward

Task: Implement

Open the file `exercise_code/networks/layer.py`. Implement the `affine_forward` and the `affine_backward` function and test your implementation by running the following cell.

```
[7]  # Test your affine layer implementations
     print(AffineTestWrapper()())
```

```
AffineForwardTest passed.
AffineBackwardTestDx passed.
AffineBackwardTestDw passed.
AffineBackwardTestDb passed.
Congratulations! You have passed all the unit tests!!! Tests passed: 4/4
Score: 100/100
You secured a score of: 100
```

Forward：計算 WX+b

```python
def affine_forward(x, w, b):
    """
    Computes the forward pass for an affine (fully-connected) layer.
    The input x has shape (N, d_1, ..., d_k) and contains a minibatch of N
    examples, where each example x[i] has shape (d_1, ..., d_k). We will
    reshape each input into a vector of dimension D = d_1 * ... * d_k, and
    then transform it to an output vector of dimension M.
    Inputs:
    :param x: A numpy array containing input data, of shape (N, d_1, ..., d_k)
    :param w: A numpy array of weights, of shape (D, M)
    :param b: A numpy array of biases, of shape (M,)
    :return out: output, of shape (N, M)
    :return cache: (x, w, b)
    """
    N, M = x.shape[0], b.shape[0]
    out = np.zeros((N,M))
    ###########################################################################
    # TODO: Implement the affine forward pass. Store the result in out.       #
    # You will need to reshape the input into rows.
    ###########################################################################

    out = np.dot(w.T, x.reshape(N, -1).T).T + b

    ###########################################################################
    #                             END OF YOUR CODE
    ###########################################################################
    cache = (x, w, b)
    return out, cache
```

Backward 計算 x、w、b 之 gradient

```python
def affine_backward(dout, cache):
    """
    Computes the backward pass for an affine layer.
    Inputs:
    :param dout: Upstream derivative, of shape (N, M)
    :param cache: Tuple of:
        - x: Input data, of shape (N, d_1, ... d_k)
        - w: Weights, of shape (D, M)
        - b: A numpy array of biases, of shape (M,
    :return dx: Gradient with respect to x, of shape (N, d1, ..., d_k)
    :return dw: Gradient with respect to w, of shape (D, M)
    :return db: Gradient with respect to b, of shape (M,)
    """
    x, w, b = cache
    dx, dw, db = None, None, None
    ###########################################################################
    # TODO: Implement the affine backward pass.
    # Hint: Don't forget to average the gradients dw and db
    ###########################################################################

    dx = np.dot(w, dout.T).T.reshape(x.shape)
    dw = np.dot(x.reshape(x.shape[0], -1).T, dout).reshape(w.shape) / x.shape[0]
    db = dout.sum(axis = 0).reshape(b.shape) / x.shape[0]

    ###########################################################################
    #                             END OF YOUR CODE
    ###########################################################################
    return dx, dw, db
```

CIFAR10 Dataset 建立與測試

```
[9] os.chdir('/content')
```

```
[10] # Define output path similar to exercise 3
     i2dl_exercises_path = os.path.dirname(os.path.abspath(os.getcwd()))
     cifar_root = os.path.join(i2dl_exercises_path, "datasets", "cifar10")

     # Dictionary so that we can convert label indices to actual label names
     classes = [
             'plane', 'car', 'bird', 'cat', 'deer',
             'dog', 'frog', 'horse', 'ship', 'truck',
     ]

     # Simply call dataset class
     dataset = ImageFolderDataset(
                 root=cifar_root
             )
```

```
Downloading https://cdn3.vision.in.tum.de/~dl4cv/cifar10.zip to /datasets/cifar10/cifar10.zip
120725504it [00:06, 19719567.34it/s]
```

Now we can set up a dataset iterate over it and visualize images as well as labels easily just like that.

```
[11] num_images = 3

     for i in range(num_images):
             item = dataset[i]
             image = item['image']
             label = item['label']

             # Print shape and label
             print('Sample {}\nimage shape: {}\nlabel: {}'.format(
                     i, image.shape, classes[label]))

             # Visualize image
             plt.subplot(1, num_images, 1 + i)
             plt.imshow(image.astype('uint8'))

     print('\nSample images')
     plt.show()
```

```
Sample 0
image shape: (32, 32, 3)
label: bird
Sample 1
image shape: (32, 32, 3)
label: cat
Sample 2
image shape: (32, 32, 3)
label: truck

Sample images
```

Task: Proof

Think about why this solves the numerical stability problem and prove that $\sigma(x) = \sigma(x + c)$ for any constant vector $c \in \mathbb{R}^n$.

With that proof, we can simply switch out the softmax computation with the new vector above and avoid numerical instabilities.

證明若將 softmax 的輸入 x 的每一項都加上一個常數後，結果會與原本相同

$$\sigma(x+c) = \frac{e^{x_i+c}}{\sum_{i=1}^{n} e^{x_i+c}} = \frac{e^{x_i} \cdot e^{c}}{\sum_{i=1}^{n} (e^{x_i} \cdot e^{c})} = \frac{e^{c} \cdot e^{x_i}}{e^{c} \sum_{i=1}^{n} e^{x_i}} = \frac{e^{x_i}}{\sum_{i=1}^{n} e^{x_i}} = \sigma(x)$$

Sanity Check

Let's quickly check if our loss formulation works as intended. Let's compute the loss of a random vector from our network defined above.

```
[17] # Set up loss
     loss_func = CrossEntropyFromLogits()

     # Sample input from a single image
     sample_image = dataset[0]['image']
     sample_label = dataset[0]['label']
     single_image_batch = np.expand_dims(sample_image, 0)
     single_label_batch = np.expand_dims(sample_label, 0)

     # Feed forward using our network
     model_output = model.forward(single_image_batch)

     # Loss computation
     computed_loss, _ = loss_func(model_output, single_label_batch)
     print('Loss of single image sample:', computed_loss)
```

```
Loss of single image sample: 2.3025235137032287
```

Task: Reason

Why do we expect our loss to be close to -log(0.1)? Explain briefly.

CE(y, ŷ)可以透過 softmax(x)模擬。又因為 softmax(x)是一個代表機率的函數(部分/全體)，所以可知 softmax(x) ≤ 1，且

$-\log(0.1) = -\log(10^{-1}) = (-1) * (-\log(10)) = \log(10) = 1$

所以可以預期 loss 會近似於-log(0.1) = 1

計算 Neural Network 的大小

Optional: Check Code

Check our implementation to compute the size of a network forward pass in bytes in `exercise_code/networks/compute_network_size.py`, which simply sums up the caches values as well as gradients. You should also think about how and why those caches/gradients are populated using the steps above.

Nice! That is the amount of memory required to do a full training forward and backward pass using our small batch.

However, if we wanted to compute the memory required to do a full gradient update for the CIFAR10 dataset using our small network, you'd need...

```
[22] # A current batch consists of 8 images. The whole dataset would require 50000/8 times the amount of memory
     num_bytes = num_bytes * 50000 / 8

     print('Total number of bytes used by network for the whole dataset', GetHumanReadable(num_bytes))
```

```
Total number of bytes used by network for the whole dataset 36.81GB
```

SGD 實作

Note: Good Practice

Always, always, always when starting a new project or defining a new network: **overfit on a small set first and then generalize**. The 500 images we are using here are already too many sample for most cases. Start with a single sample, then 10 and finally a few hundred. Don't cheap out on this step! More often, your network will fail to generalize properly and you have to first know if it has enough capacity to overfit and that the full training pipeline is working!

In order to run these experiments, you don't necessarily need a validation set. Just a few training samples are enough to make those checks!

```
[23] # Redefine model and loss function
     model = ClassificationNet(input_size=input_size,
                                                    hidden_size=128,
                                                    activation=Relu(),
                                                    num_layer=2,
                                                    num_classes=10)

     loss_func = CrossEntropyFromLogits()
```

```
[24] learning_rate = 1e-2

     # We use our training dataloader for validation as well as testing
     solver = Solver(model, dataloader, dataloader,
                                  learning_rate=learning_rate, loss_func=loss_func, optimizer=SGD)

     # This might take a while depending on your hardware. When in doubt: use google colab
     solver.train(epochs=20)

     (Epoch 1 / 20) train loss: 2.302592; val loss: 2.302594
     (Epoch 2 / 20) train loss: 2.302559; val loss: 2.302325
     (Epoch 3 / 20) train loss: 2.302292; val loss: 2.302050
```

SGD + Momentum 實作：藉由更新 velocity 來計算 learning rating

4.3 SGD + Momentum

As you can see, the loss is going down smoothly which indicates that we are easily overfitting. Great. However, plain SGD is rarely used in practice (as it is usually too slow) which is why we will focus on implementing SGD+Momentum now, which is a straightforward extension to SGD.

Recall that its update rule is defined by:

$$v^{k+1} = \beta v^k - \alpha \nabla_\theta L(\theta^k),$$
$$\theta^{k+1} = \theta^k + v^{k+1}.$$

Task: Check Code and Implement

Familiarize yourself with the SGD implementation in `exercise_code/networks/optimizer.py` as well as our general optimization class structure.

Then, implement the `SGDMomentum._update` function which is very similar to the update rule of SGD above.

```
[25] #Test your SGD momentum implementations
     print(SGDMTestWrapper()())

     SGDM_Weight_Test passed.
     SGDM_Velocity_Test passed.
     Congratulations you have passed all the unit tests!!! Tests passed: 2/2
     Score: 100/100
     You secured a score of :100
```

為了以防原先 config[“momentum”]為空，先將 0.9 加入字典中

```python
    def _update(self, w, dw, config, lr):
        """
        Update a model parameter
        """
        if config is None:
            config = {}
        config.setdefault('momentum', 0.9)
        v = config.get('velocity', np.zeros_like(w))
        next_w = None

        ########################################################################
        # TODO: Implement the momentum update formula. Store the updated value in
        # the next_w variable. You should also use and update the velocity v.
        ########################################################################

        b = config.get('momentum')
        v = b * v - lr * dw
        next_w = w + v

        ########################################################################
        #                                                    END OF YOUR CODE
        ########################################################################
        config['velocity'] = v

        return next_w, config
```

測試與比較 SGD、SGD+Momentum、Adam

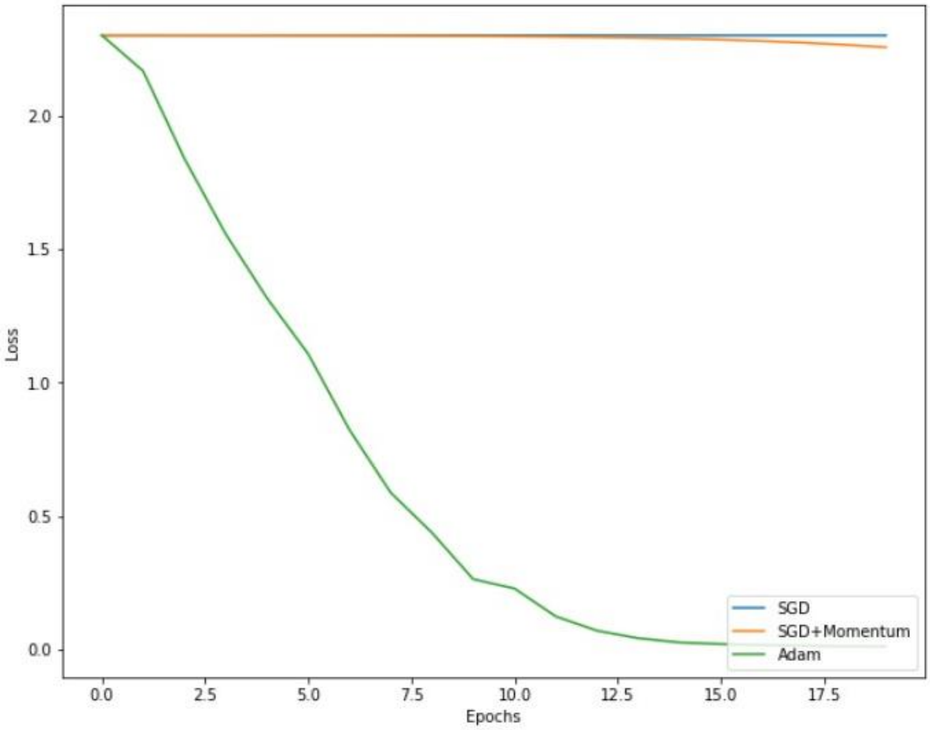
```python
[26] learning_rate = 1e-3
num_epochs = 20
loss_func = CrossEntropyFromLogits()

# Compute loss histories for all optimizers
loss_histories = {}

for name, optimizer in zip(['SGD', 'SGD+Momentum', 'Adam'], [SGD, SGDMomentum, Adam]):
    print('Starting {}'.format(name))
    # Reset model
    model = ClassificationNet(input_size=input_size,
                              hidden_size=128,
                              activation=Relu(),
                              num_layer=2,
                              num_classes=10)
    # Set up solver
    solver = Solver(model, dataloader, dataloader,
                    learning_rate=learning_rate, loss_func=loss_func,
                    optimizer=optimizer)
    solver.train(epochs=num_epochs)
    # Save train history to plot later
    loss_histories[name] = solver.train_loss_history
    print()

# Plot them in a shared plot
for name in loss_histories:
    plt.plot(loss_histories[name], '-', label=name)
plt.legend(loc='lower right')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```

比較結果：



存檔與繳交

7. Submission Instructions

Hoooooooray, you trained your model! The model will be saved as a pickle file to `models/NN.p`.

```
[27] from exercise_code.tests import save_pickle
     from exercise_code.networks.layer import *
     from exercise_code.networks.optimizer import SGDMomentum

     save_pickle(
             data_dict={
                     "SGD_Momentum_update": SGDMomentum._update,
                     "AffineForward": affine_forward,
                     "AffineBackward": affine_backward,
                     "Sigmoid": Sigmoid,
                     "Relu": Relu,
             },
             file_name="NN.p"
     )
```

```
[28] from exercise_code.submit import submit_exercise


     submit_exercise('exercise05')
```

```
relevant folders: ['models']
notebooks files: []
Adding folder models
Zipping successful! Zip is stored under: /content/exercise05.zip
```